intel®

APPLICATION NOTE

AP-182

July 1984

Multimodule™ Winchester Controller Using the 82062

J. SLEEZER
TECHNICAL MARKETING

Order Number: 231133-002

MULTIMODULE™ WINCHESTER CONTROLLER USING THE 82062

CONTENTS

CONTENTS

PAGE

CONTENTS

PAGE

INTRODUCTION

The 82062 Winchester Disk Controller (WDC) was developed to ease the complex task of interfacing Winchester disk drives to microprocessor systems. Specifically, the 82062 WDC interfaces to drives that conform to the ST506 specification, which is the dominant interface for $5\frac{1}{4}$ inch drives. This Application Note provides some background on the 82062 WDC, the drive interfaces and general software routines. It concludes with a design example using the 82062 WDC interfaced to the SBX™ bus. Appendix B contains the listing of the software necessary to operate this controller board.

ST506 Winchester Drive Overview

Since the 82062 WDC interfaces only to drives conforming to the ST506 specification, this overview will limit itself to those drives. A summary of the ST506 specification is shown in Appendix A for those who are not familiar with it. The ST506 Winchester Disk contains from 1 to 8 hard disks (or platters) with the average being 2 to 3 disks. These disks are made from aluminum (hence the term hard disk) and are coated with some type of recording media. The recording media is typically made of magnetic-oxide, which is similar to the material used on floppy disks and cassette tapes. Each side of a hard disk is coated with recording media and each side can store data. Each surface of a disk has its own read/write head.

Hard disk drives are sealed units because the R/W heads actually fly above the disk surface at about 8 to 20 microinches. A piece of dust or dirt, which appears as a boulder to the gap between the heads and the disk surface, will wreak havoc upon the disk media.

The R/W heads are mechanically connected together and move as a single unit across the surface of the disk. There are 2 basic methods for positioning the heads. The first is with stepper motors, which is the most common method and is also used on most floppy disk drives. These positioners are used mainly because of their low cost.

The second method of positioning the heads is to use a voice-coil mechanism. These units do not move in steps but swing across the disk. These mechanisms generally permit greater track density than steppers, but also require complex feedback electronics which increases the cost of the drive. Generally, voice-coil head positioners use closed loop servo positioning, as compared to the open loop positioning used with stepper motors.

The surface of a disk is divided logically into concentric circles radiating from the center as shown in Figure 1. Each concentric circle is called a track.

The group of same tracks on all cylinders is collectively called a cylinder. The number of tracks on a surface (which affects storage density) is determined by the head positioners. Typically, stepper head positioners have fewer tracks than drives that use a voice coil positioner. Which type of positioner is used is irrelevant to the 82062 as positioners are part of the drive electronics. The 82062 can access up to 1024 tracks per surface.

Once the surface is divided into cylinders it is further divided radially (as with a pie). The area between the radial spokes is referred to as a sector. The number of sectors per track is determined by many variables, but is basically determined by the number of data bytes and the length of the ID field (which locates a sector). Figure 2 shows one manufacturer's specifications for their drive. The manufacturer formats the drive with 32–256 byte sectors per track. Alternatively, the drive could be reformatted to contain 17–512. byte sectors per track. This second option has fewer sectors per track but stores more data. Determining how many bytes each sector contains is done by extensive analysis of the hardware and operating system. The 82062 WDC is programmable for sector size during formatting.

The order in which sectors are logically numbered on the track is called interleaving. An interleave factor of four would have three sectors separating logically se-

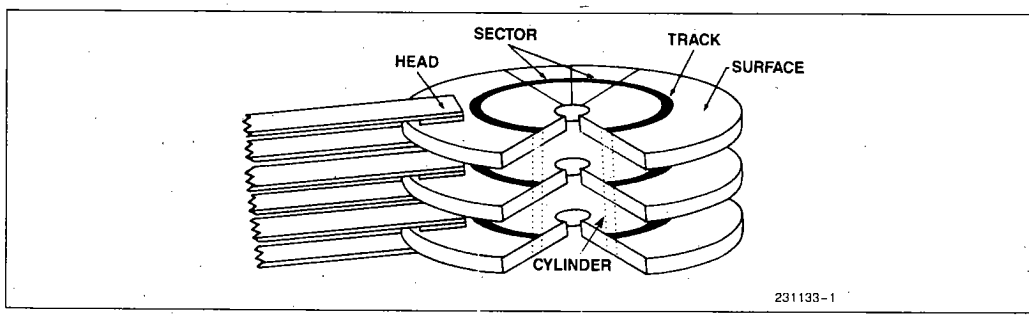


Figure 1

Capacity	
Unformatted	
Per Drive	6.38 Megabytes
Per Surface	1.59 Megabytes
Per Track	10416 Bytes
Formatted	
Per Drive	5.0 Megabytes
Per Surface	1.25 Megabytes
Per Track	8192 Bytes
Per Sector	256 Bytes
Sectors per Track	32
Transfer Rate	5.0 Megabits per second
Access Time	
Track to Track	3 ms
Average (Inc. Settle)	170 ms
Maximum (Inc. Settle)	500 ms
Settling Time	15 ms
Average Latency	8.33 ms
Functional Specifications	
Rotational speed	3600 rpm ± 1%
Recording density	7690 bpi max
Flux density	7690 fci
Track density	255 tpi
Cylinders	153
Tracks	612
R/W Heads	4
Disks	2

Figure 2. A Typical Drive Specification

quential sectors. Starting at the index pulse, an example of four way interleaving is:

Sector 1, Sector X, Sector Y, Sector Z, Sector 2, Sector . . .

Interleaving is used primarily because one sector at a time is transferred from disk to sector buffer to system RAM. Without interleaving, the delay in transferring data would result in sectors on the disk rotating past the heads. The operating system would then have to wait one disk revolution to get to the next sector (a 16.7 msec delay). With interleaved sectors, the next logical sector would be positioned beneath the heads after the previous sector of data had been transferred to the system RAM. Interleaving unfortunately slows down the overall transfer rate from the disk. A 5 Mbit/second transfer rate averages out to a 1.25 Mbit/second transfer rate when many sectors are transferred with four way interleaving. Again, how much interleaving to use is determined by extensive hardware/software benchmarking.

Whenever data is stored on a multiple platter disk drive, the same track on all surfaces whould be used before repositioning the heads to another track. Repositioning the heads generates a longer delay due to the mechanical delay of moving the heads. Switching to another head incurs no mechanical positioning delay. Only one head can be selected at a time.

Hard disk drives tend to be faster than floppies for two reasons. The speed at which the disk spins is about 10 times faster than the floppy (a floppy spins at 360 rpm). This yields an immediate one-tenth reduction in access times for the same size drive. While both ST506 drives and floppies use stepper motors, the steppers utilized by the hard disk drives are approximately twice as fast as those used by floppies.

82062 WINCHESTER DISK CONTROLLER

The 82062 WDC provides most of the functions necessary to interface between a microprocessor and an ST506 compatible disk drive. The 82062 converts the high level commands and parallel data of a microprocessor bus into ST506 compatible disk control signals and serial MFM encoded data. This section presents a detailed description of the 82062 and a discussion of various techniques which can be used to interface the 82062 to a microprocessor.

The internal structure of the 82062 is divided into several sections as shown in Figure 3. They are:

1. the microprocessor interface which includes the status and task registers;
2. sector buffer control;
3. the drive interface;
4. and the data transfer section, which includes the CRC logic and the conversion and MFM encoding/decoding of microprocessor data.

Clock Inputs

The 82062 has two clock inputs: read clock (RD CLOCK) and write clock (WR CLOCK). The PLA controller, the processor interface, buffer control and MFM encoding sections operate off the WR CLOCK input. The RD CLOCK input is used only for decoding the MFM data stream. The clocks may be asynchronous to one another. Both clocks have non-TTL compatible inputs. The easiest method to interface to TTL requires a pull-up resistor to satisfy their input voltage needs. The resistor's value must be compatible with the VIL specification of these pins. See the Pin Descriptions Section for more specific information.

Microprocessor Interface

The microprocessor interface of the 82062 contains the control logic which permits commands and data to be

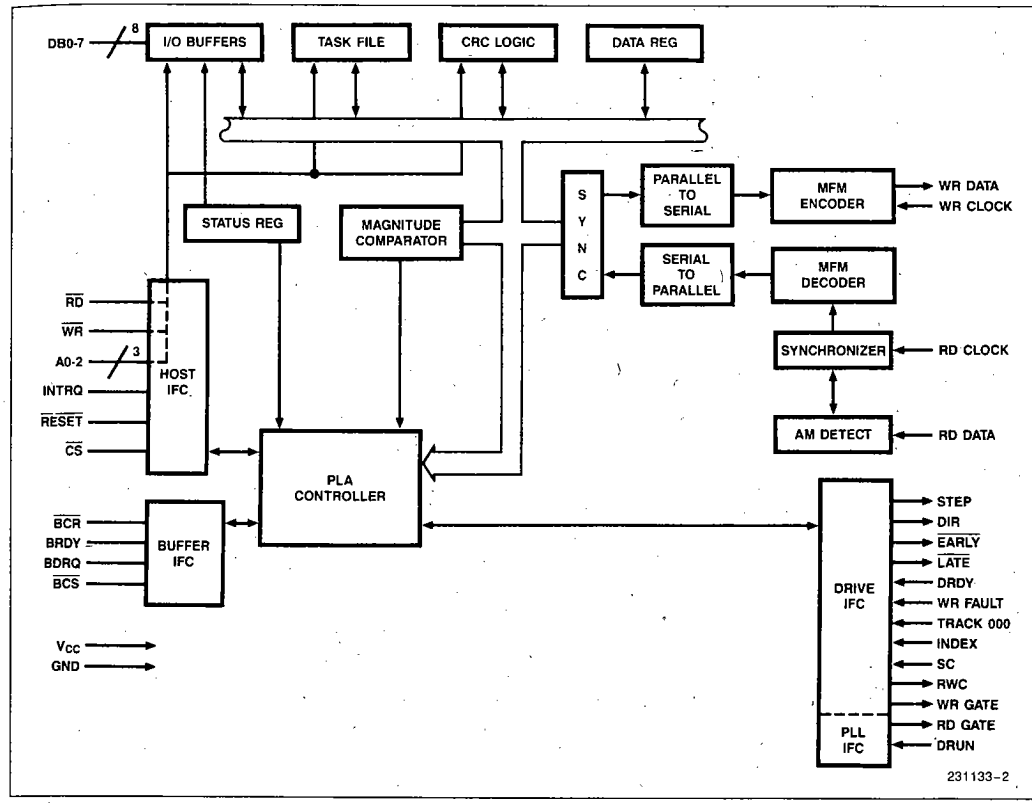


Figure 3. 82062 Internal Block Diagram

transferred between the host and the 82062. The interface consists of an 8 bit, tri-state, bidirectional data bus; the task registers; a 3 to 8 address decoder for selecting one of the seven registers; and the general read, write, and chip select logic. Externally, the 82062 expects a buffer equal in size to a sector on the disk, and tri-state transceivers between the sector buffer and the microprocessors data bus in order to isolate itself from the microprocessor during disk data transfers.

A0–A2, Data Bus

These three address lines are active high signals and select one of the seven register locations in the 82062. They are not latched internally. If the three addresses are equal to 0 and the 82062 is selected, the data bus is kept tri-stated to ease interfacing to a sector buffer. The 82062's data bus is controlled by both the microprocessor and the 82062. The microprocessor has control for loading the registers and command. During disk reads or writes, control switches to the 82062 so that it may access the local sector buffer when transferring data between the disk and the buffer.

$\overline{RD}$, $\overline{WR}$, $\overline{CS}$

The chip select ($\overline{CS}$) is typically decoded from the higher order address lines. $\overline{CS}$ only permits data to be placed into, or read from, the 82062's task registers. Once a disk operation starts, $\overline{CS}$ no longer efffects the 82062. $\overline{RD}$ and $\overline{WR}$ are bidirectional lines and are used to read or write the 82062's registers by the host microprocessor and are valid only if $\overline{CS}$ is present. The 82062 will drive $\overline{RD}$ and $\overline{WR}$ when transferring data between the sector buffer and the disk. A signal is provided to tri-state the $\overline{RD}$ and $\overline{WR}$ lines from the host during a buffer access. This is covered in the Sector Buffer Control Section.

Interrupts

An interrupt is issued at the end of all commands, and the interrupt is cleared by reading any register. For the Read Sector command only, the 82062 allows the user the option of an interrupt either at the termination of the command, as is the case with all other commands,

or when data needs to be transferred to the host from the sector buffer. This is discussed further in the Interrupt Mode Section. When selecting the data transfer option, the interrupt line will go active at the same time as the BDRQ line and the interrupt will be removed only when the proper handshake occurs with the sector buffer.

Task Registers

The Task Register File contains the command, status, track number, sector number, and other information necessary to properly execute a command. These registers are accessed with A0–A2, $\overline{RD}$ (or $\overline{WR}$), and $\overline{CS}$ being valid and are not cleared by a reset. The registers are covered in detail in the Task Register File Section.

Sector Buffer Control

The 82062 was designed to operate with an external buffer equal in size to one sector. To ease the design-in of this buffer, the 82062 provides all of the control signals it needs to operate the buffer. This buffer must be isolated from the system bus, using tri-state buffers, during disk transfers to prevent contention during the period that the 82062 is accessing the buffer. A sector buffer is generally used to ease interfacing to the system due to the high disk data rates (625 kbytes/sec), although it is not required.

$\overline{BCS}$

The Buffer Chip Select ($\overline{BCS}$) line goes active whenever the 82062 is accessing the sector buffer. This signal should remove the microprocessors ability to access the 82062 and sector buffer and must enable the sector buffer for use by the 82062.

At a 5 Mbit/sec disk data rate, the 82062 will access the buffer every 1.6 microseconds (8 bits × 200 ns/bit). $\overline{BCS}$ will remain low the entire time the 82062 is accessing the buffer. The 82062 will pulse the appropriate $\overline{RD}$ or $\overline{WR}$ line for each byte transferred.

$\overline{BCR}$

Buffer Counter Reset ($\overline{BCR}$) goes active each time that $\overline{BCS}$ changes state. Its purpose is to reset the address counter of the sector buffer back to zero before and after the 82062 uses the sector buffer. Its function is optimized for single sector transfers. Multiple sector transfers should use a software controlled buffer counter reset and not use $\overline{BCR}$ as the sector buffer will be reset to the beginning after each sector is transferred.

BDRQ, BRDY

Buffer Data Request (BDRQ) and Buffer Ready (BRDY) provide the handshake needed to transfer data between the sector buffer and the host. BDRQ signals that data must be moved to/from the sector buffer and the host. BRDY has two functions. Once the transfer signaled by BDRQ is finished, asserting BRDY will inform the 82062 that the transfer is completed and that it may finish executing the command. BRDY is also used in multiple sector commands. BRDY going high during a multiple sector transfer indicates that the buffer is full (or empty—depending upon the command) and the transfer should wait until the buffer is serviced. The sector that was being transferred will finish and the 82062 will deactivate $\overline{BCS}$ and activate BDRQ. The host microprocessor must then transfer the data between the buffer and system memory. When this transfer is finished, asserting BRDY will cause the 82062 to resume the command.

The handshaking between BDRQ and BRDY occurs only in full sector increments—not on a byte basis. A high on BDRQ indicates a full sector's worth of data is required; BRDY going high indicates a full sector of data is available to the 82062 without interruption.

Only the rising edge of BRDY is valid. A falling edge may occur at any time without effect. $\overline{BCR}$ will pulse and $\overline{BCS}$ will go active eight byte times (8 bytes × 8 bits/byte × 200 ns/bit = 12.8 microseconds) before the first data byte is transferred from the sector buffer to the disk.

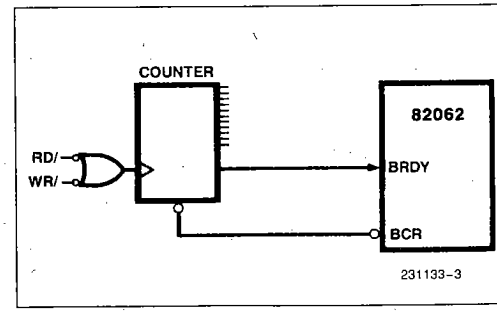


Figure 4. BRDY Generation Logic

Data Transfer Logic

This section of the 82062 is responsible for conversion of serial disk data to parallel data (and vice versa); encoding/decoding of the disk's MFM serial bit stream; detecting the address mark; and verifying data integrity through its CRC generation and checking logic.

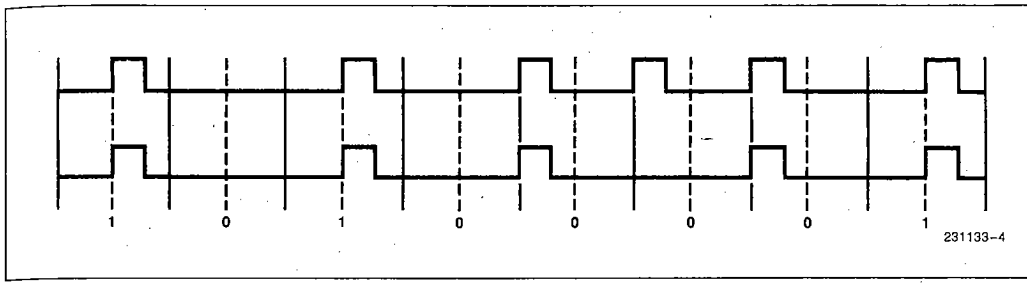


Figure 5. Data Address Mark

MFM Encoding/Decoding

The MFM encoding section will receive 8 bit parallel data when a valid command has been recognized and BRDY has gone high. The parallel data is first serialized and converted to an intermediate, NRZ encoded, data stream. The serial NRZ data is sent to the MFM encoding section and then transferred to the disk. Decoding of the MFM bit stream (during disk reads) happens in reverse order.

The control logic operates off the write clock (WR CLOCK) running at a frequency of the desired tranfer rate. The MFM decoding portion operates off of the read clock (RD CLOCK) input, which is supplied by an external phase lock loop. The two clocks need not be synchronized to each other. Data is written (and hence read) with the most significant bit first.

Address Mark Detector

The address mark is a unique 2 byte code written at the beginning of each ID field and data field. This address mark serves two purposes. It tells the controller what type of data is about to be received so that internal computations can be performed, and to ensure that ID fields are not sent to the host. The second purpose is to align the serial data back to the original 8 bit boundaries that existed when data was written (there are no byte boundaries on a disk).

An address mark is always preceded by the all zeros synchronization field. The 82062 starts comparing the incoming data stream when the synchronization field ends. A high speed comparator is used since the 82062 does not yet know where the proper byte boundaries are. When a proper comparison of the address mark is made the controller starts assembling bytes, starting with the second byte of the address mark.

The first byte of the address mark is an "A1" Hex, but purposely violates the MFM encoding rules by removing a clock pulse. In Figure 5, the first example is of a normal MFM encoded A1H. The second example is of the address mark and shows the missing clock pulse. The non-MFM compatible A1 is to prevent the host from issuing a similar data byte and possibly confusing detection logic.

The second byte specifies either an ID or data field and is encoded according to normal MFM rules. It is either an "F8" Hex for a data field, or "FC" through "FF" for an ID field. The different values correspond to a range of cylinders on the drive in increments of 256 tracks. The 82062 makes no use of this information, but writes it for compatibility with the ST506 specification during formatting.

CRC Generation/Checking

The CRC generator computes and checks the cyclic redundancy check bytes that are appended to the ID and data fields. CRC generation/checking is always done on ID fields. Data fields have a choice between 82062 CRC or externally supplied ECC. Read Sector commands with a CRC error will still have transferred the data into the sector buffer. When bit 7 in the SDH register is low (enabling CRC for data fields) the CRC bytes are not transferred to the sector buffer or host.

The generator polynomial for the CRC-CCITT (CRC-16) code is:

$$x16 + x12 + x5 + 1 = (x + 1)(x15 + x14 + x13 + x12 + x4 + x3 + x2 + x + 1)$$

The code's capability is as follows:

a) Detects all occurrences of an odd number of bits in error.

b) Detects all single, double, and triple bit errors if the record length (including check bits) is less than 32,767 bits.

c) Detects all single-burst errors of sixteen bits or less.

d) Detects 99.99695% of all possible 17 bit burst errors, and 99.99847% of all possible longer burst, assuming all errors are possible and equally probable.

The CRC code has some double-burst capability when used with short records (sectors). For a 256 byte sector the code will detect double-bursts as long as the total number of bits in error does not exceed 7.

PLA Control

The PLA Controller interprets command sent by the microprocessor. Its operation is synchronized to the WR CLOCK input. The PLA controller is started when a command is written into the command register. It generates control signals and operates in a handshake mode when communicating with the MFM decoding block.

Magnitude Comparator

A 10 bit magnitude comparator is used to calculate the direction and number of step pulses needed to move the head from the present cylinder position to the desired position. A separate high speed equivalence comparator is used to compare ID field bytes when searching for a sector ID field.

Drive Interface

The drive interface of the 82062 contains the logic that makes possible the storage and reliable recovery of data. This interface consists of the drive and head select logic, the disk control signals, and read and write data logic as shown in Figure 6. This section describes the external circuitry which is required to complete the 82062's drive interface.

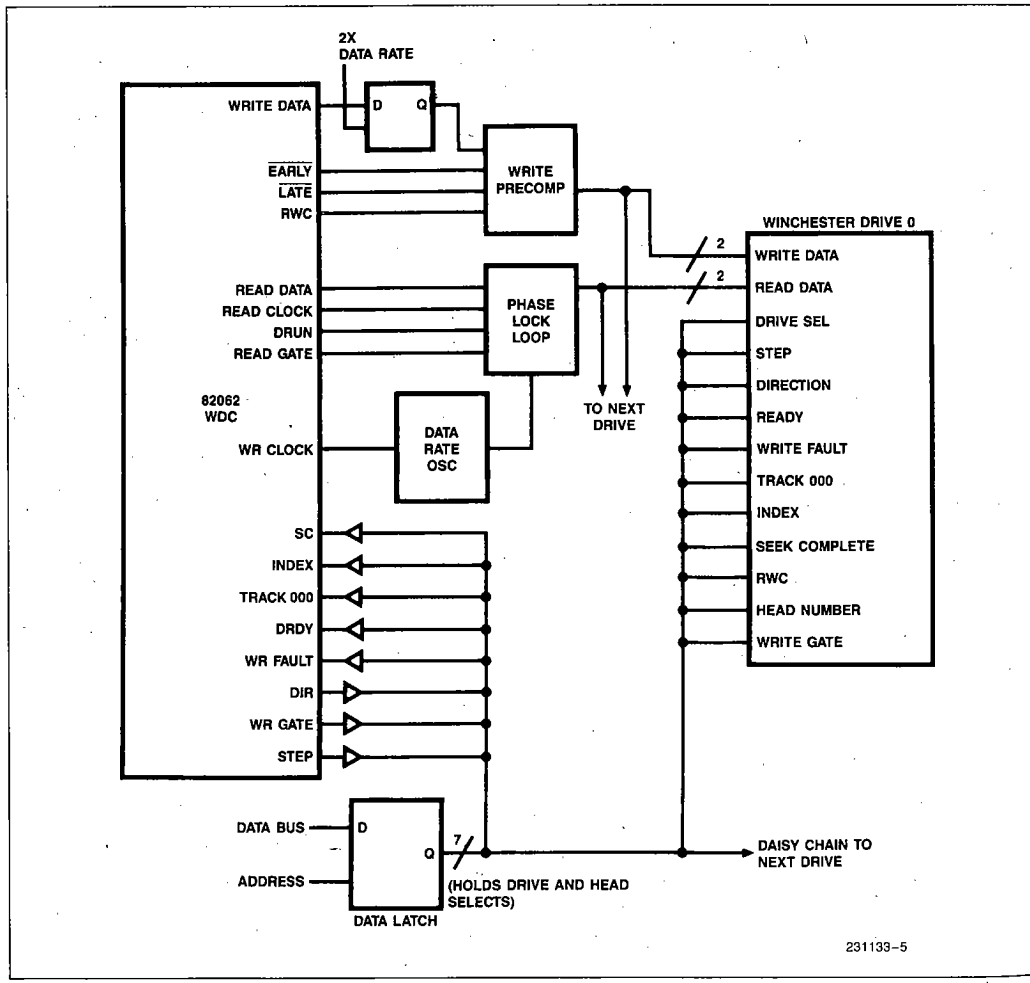


Figure 6. Drive Interface

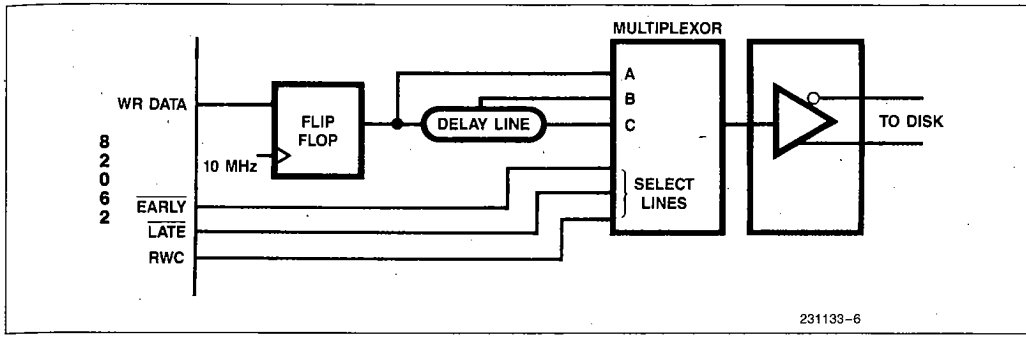


Figure 7. Write Precompensation Logic

Drive/Head Select

The 82062 has no outputs for selecting the head or drive. Therefore these signals must be generated by the user as shown in Figure 6. Data bits 0–4 should be latched whenever the SDH register is written. Bits 0–2 would then be driven onto the drive cable with open collector buffers. Bits 3 and 4 would be decoded after being latched, then buffered for the cable. The head information written to the 82062's SDH register is used to write the proper ID fields during formatting. Changing the drive bits in the SDH register will cause a Scan ID to be performed by the 82062 to update non user accessible registers.

Drive Control

The drive control (STEP, DIR, WR FAULT, TRACK 000, INDEX, SC, RWC, and WR GATE) signals are merely conditioned for transmission over the drive cable. The purpose of each pin can be found in the section on Pin Descriptions and their use in the Command Section.

WR DATA, $\overline{\text{EARLY}}$, $\overline{\text{LATE}}$

Figure 7 is a diagram of the interface required on the write data line. The final stage of the MFM encoding requires applying the WR DATA to an external flip-flop clocked at 10 MHz. The 82062 monitors the serial write data output for particular bit patterns that require precompensation to prevent bit shifting. $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ are active on all cylinders and will normally require that RWC be factored into them to activate the data precompensation on the proper cylinder.

A delay line is required to generate the delayed data for precompensation since the actual delay varies between drive manufacturers. $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ go active in the same clock period that generates the data bit to be shifted.

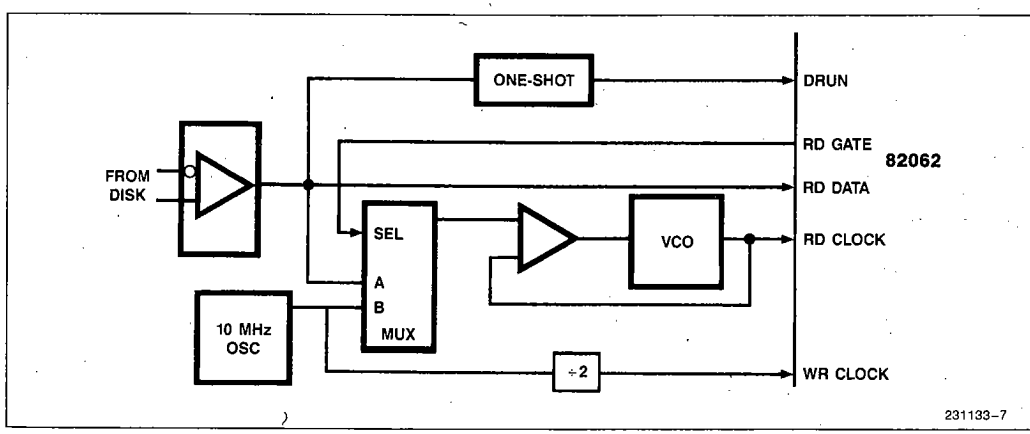


Figure 8. Data Recovery Logic

RD Data, DRUN, RD Gate

The read data interface is shown in Figure 8, and consists of the data run (DRUN) signal and a phase lock loop to generate the RD CLOCK input to decode the serial data. DRUN is generated from a retriggerable one-shot with a period just exceeding one bit cell. A sync field consisting of a string of clock pulses will continually retrigger the one-shot producing a steady high level on DRUN. The 82062 counts off 16 clock pulses internally, and if DRUN is still active, will make RD GATE active. Any byte other than an address mark will deactivate RD GATE and the sequence starts over.

The phase lock loop generates RD CLOCK which is used to decode the incoming serial data. Until RD GATE is activated by the 82062, the phase lock loop (PLL) should be locked onto a local 10 MHz clock to minimize PLL lock-up times. When RD GATE is activated, the PLL starts locking onto the incoming data stream, which should consist of the all zeros sync field. Once the PLL locks onto this synch field, the 82062 will start examining the serial data for a non-zero byte. A non-zero byte will be indicated by DRUN dropping since the address mark follows the sync field and is an "A1" Hex. This sequence is shown in Figure 9. If the address mark is detected, and if it was preceded by at least 9 bytes of zeros, RD GATE will stay active. The 82062 will then assemble bytes of data, and ensure the proper ID field is found. If a non-zero or non-address mark byte was detected, RD GATE will go inactive for a minimum of 2 byte times. If a data field or the wrong ID field is detected, or the ID field was not preceded by 8 bytes of zeros, then RD GATE goes inactive and the sequence starts over with the 82062 examining the DRUN input.

Microprocessor Interfaces

This section shows the general 82062 interfaces to a microprocessor system. There are essentially four interfaces which consist of a combination of polled, DMA, and interrupts. While the 82062 was designed to interface directly to one type, it accommodates all with minor additional logic.

DMA Interface

The 82062 is designed to use a DMA controller for data transfer between its sector buffer and the host system, and to interrupt the host when the command has finished. This interface is shown in Figure 10.

When the 82062 determines that a transfer is needed between the sector buffer and the host (either at the beginning of a command or through BRDY going active in a multiple sector transfer), it will assert BDRQ. BDRQ will initiate a DMA transfer via the DMA request input. The DMA controller will generate reads or writes which will increment an address counter. BRDY indicates that the data transfer has finished and is issued off the carry-out line (or high order address line) of the counter. The 82062 will assert BDRQ at this point and activate $\overline{BCS}$ to prevent the host from intefering with disk/buffer transfers. There can be no polling for a data transfer or a register read without an interrupt in this scheme.

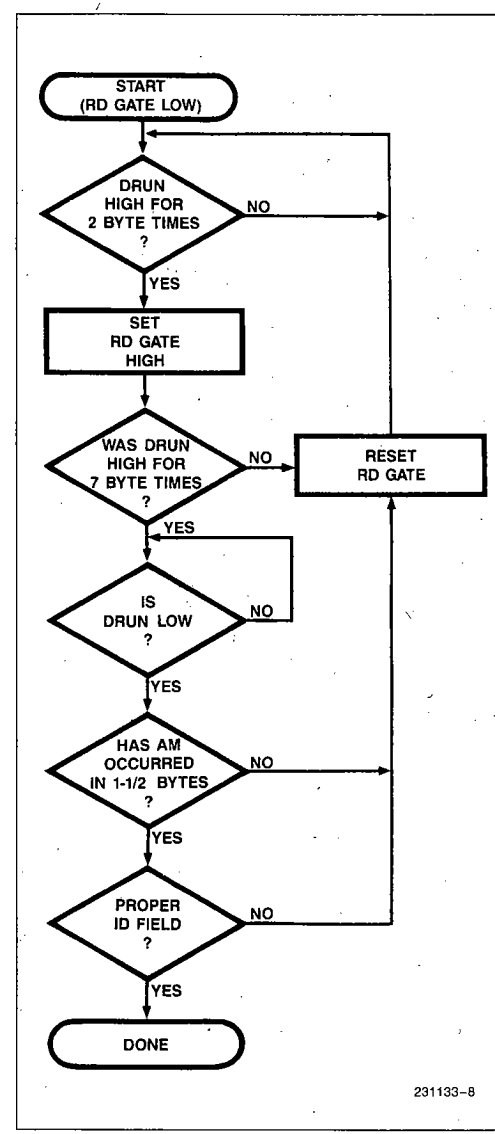


Figure 9. PLL Control Sequence

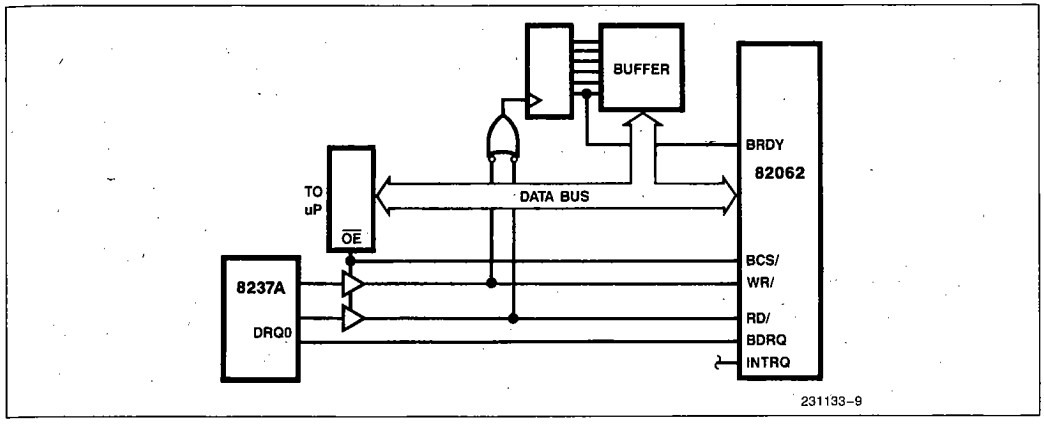


Figure 10. 82062 DMA Interface

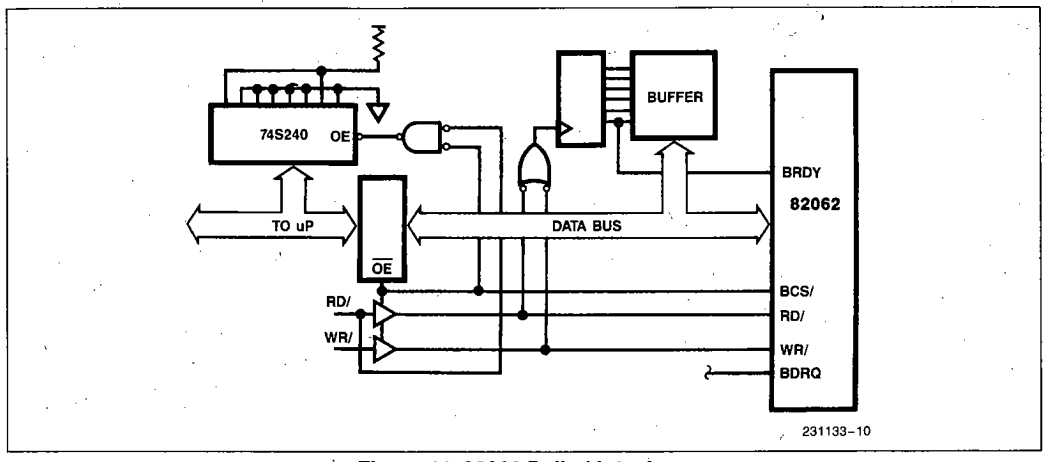


Figure 11. 82062 Polled Interface

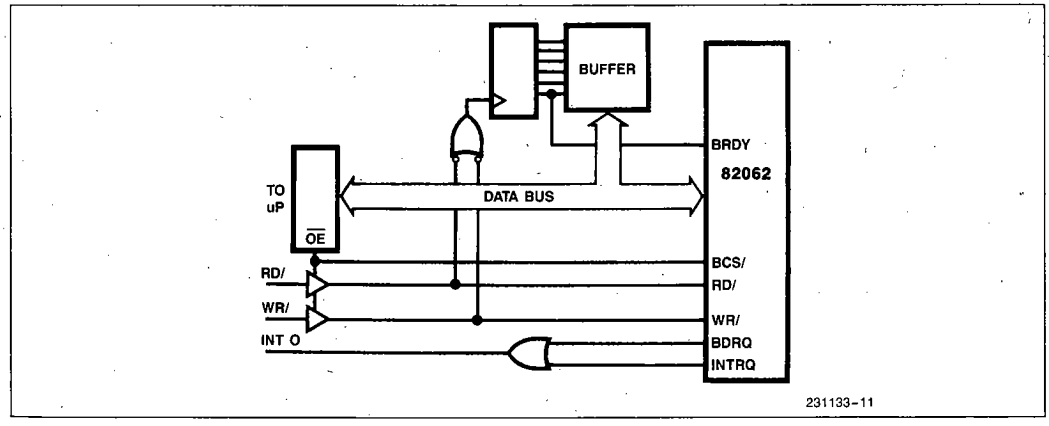


Figure 12. 82062 Interrupt Interface

Polled Interface

Since the 82062 isolates itself from the host during several commands, the host cannot read the status register during some periods to determine what course should be taken. In Figure 10, trying to read the status register when $\overline{BCS}$ is active will return indeterminate data. To prevent the microprocessor from reading this indeterminate data, a hardware generated "Busy" pattern should be driven onto the data bus if $\overline{BCS}$ is active. This is shown in Figure 11. The status register contains a data request (DRQ) bit whose timing is equal to the BDRQ output signal, thus making a polled operation possible. DRQ will stay set in the status register until a BRDY is generated.

One design issue with the polled interface occurs when the microprocessor is polling the status and the 82062 deactivates $\overline{BCS}$. The microprocessor would normally read the hardware busy pattern. If $\overline{BCS}$ is deasserted, the hardware pattern is disabled and the microprocessor will start to read the real status register. The read cycle may almost be finished, and the read access period of the 82062 will not be satisfied. The data returned to the microprocessor will be invalid.

Interrupt Interface

There are cases where the designer does not want to tie up the microprocessor with polling. The typical 82062 design will need two interrupts per command. One for a data transfer and one for the completion of the command. The 82062 has an output to issue an interrupt when the command has finished. However for data transfers an interrupt must be generated from the BDRQ line as shown in Figure 12 (whether a DMA controller is used or not). When a data transfer is needed, the 82062 will activate the BDRQ line. The microprocessor will be interrupted and do the data transfer function. BDRQ will stay active until BRDY is generated, so the system must either use edge triggered interrupts or must not write the end-of-interrupt byte until BDRQ is removed (this is true of Intel's 8259A).

PIN DESCRIPTIONS

Symbol	Pin. No.	Type	Name and Function
$\overline{BCS}$	1	O	**Buffer Chip Select:** Output used to enable reading or writing of the external sector buffer by the 82062. When low, the host should not be able to drive the 82062 data bus, $\overline{RD}$, or $\overline{WR}$ lines.
$\overline{BCR}$	2	O	**Buffer Counter Reset:** Output that is strobed by the 82062 prior to read/write operation. This pin is strobed whenever $\overline{BCS}$ changes state. Used to reset the address counter of the buffer memory.
INTRQ	3	O	**Interrupt Request:** Interrupt generated by the 82062 upon command termination. It is reset when any register is read. Optionally signifies when a data transfer is required on Read Sector commands.
N/C	4		No connection. Reserved for future use.
$\overline{RESET}$	5	I	**Reset:** Initializes the controller and clears all status flags. Does not clear the Task Registers.
$\overline{RD}$	6	I/O	**Read:** As an input, $\overline{RD}$ controls the transfer of information from the 82062 registers to the host. $\overline{RD}$ is an output when the 82062 is reading data from the sector buffer ($\overline{BCS}$ low).
$\overline{WR}$	7	I/O	**Write:** As an input, $\overline{WR}$ controls the transfer of command or task information into the 82062 registers. $\overline{WR}$ is an output when the 82062 is writing data to the sector buffer ($\overline{BCS}$ low).
$\overline{CS}$	8	I	**Chip Select:** Enables $\overline{RD}$ and $\overline{WR}$ as inputs for access to the Task Registers. It has no effect once a disk command starts.
A0–A2	9–11	I	**Address:** Used to select a register from the task register file.
DB0–DB7	12–19	I/O	**Data Bus:** Bidirectional 8-bit Data Bus with control determined by $\overline{BCS}$. When $\overline{BCS}$ is high the microprocessor has full control of the data bus for reading and writing the Task Registers. When $\overline{BCS}$ is low the 82062 controls the data bus to transfer data to or from the buffer.

Pin Descriptions (continued)

Symbol	Pin. No.	Type	Name and Function
GND	20		Ground.
WR DATA	21	O	**Write Data:** Open drain output that shifts out MFM data at a rate determined by Write Clock. Final stage requires an external flip-flop clock at 10 MHz. See note 1.
$\overline{\text{LATE}}$	22	O	**Late:** Open drain output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders. See note 1.
$\overline{\text{EARLY}}$	23	O	**Early:** Open drain output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders. See note 1.
WR GATE	24	O	**Write Gate:** High when write data is valid. WR GATE goes low if the WR FAULT input is active. This output is used by the drive to enable head write current.
WR CLOCK	25	I	**Write Clock:** Clock input used to derive the write data rate. Frequency −5 MHz for the ST506 interface, 4.34 MHz for the SA 1000 interface. See Note 2.
DIR	26	O	**Direction:** High level on this output tells the drive to move the head inward (increasing cylinder number). The state of this signal is determined by the 82062's internal comparison of actual cylinder location vs desired cylinder.
STEP	27	O	**Step:** Provides 8.4 microsecond pulses to move the drive head to another cylinder at a programmable frequency.
DRDY	28	I	**Drive Ready:** If DRDY from the drive goes low, the command will be terminated.
INDEX	29	I	**Index:** Signal from the drive indicating the beginning of a track. It is used by the 82062 during formatting, and for counting retries. Index is edge triggered. Only the rising edge is valid.
WR FAULT	30	I	**Write Fault:** An error input to the 82062 which indicates a fault condition at the drive. If WR FAULT from the drive goes high, the command will be terminated.
TRACK 000	31	I	**Track Zero:** Signal from the drive which indicates that the head is at the outermost cylinder. Used by the Restore command.
SC	32	I	**Seek Complete:** Signal from the drive indicating to the 82062 that the drive head has settled and that reads or writes can be made. SC is edge triggered. Only the rising edge is valid.
RWC	33	O	**Reduced Write Current:** Signal goes high for all cylinder numbers above the value programmed in the Write Precomp Cylinder register. It is used by the precompensation logic and by the drive to reduce the effects of bit shifting.
DRUN	34	I	**Data Run:** This signal informs the 82062 when a field of ones or zeros has been detected by an external one-shot. This indicates the beginning of an ID field. RD GATE is brought high when DRUN is sampled high for 16 clock periods. See Note 2.
BRDY	35	I	**Buffer Ready:** Input used to signal the controller that the buffer is ready for reading (full), or writing (empty), by the host μP. Only the rising edge indicates the condition.

Pin Descriptions (continued)

Symbol	Pin. No.	Type	Name and Function
BDRQ	36	O	**Buffer Data Request:** Activated during Read or Write commands when a data transfer between the host and the 82062's sector buffer is required. Typically used as a DMA request line, or to generate an interrupt.
RD DATA	37	I	**Read Data:** Single ended input that accepts MFM data from the drive. See note 2.
RD GATE	38	O	**Read Gate:** Output that is high for data and ID fields. Goes active when DRUN has been high for 16 WR CLOCK periods to permit the external phase lock loop to lock onto the incoming disk data stream.
RD CLOCK	39	I	**Read Clock:** Clock input derived from the external data recovery circuits. See note 2.
V_{CC}	40	I	**D.C. Power:** +5V

NOTES:
1. This pin requires a pull-up resistor to function properly. A value of 1000 ohms will work satisfactorily.
2. This pin requires input levels that are not TTL compatible. These lines can be interfaced to TTL with a pull-up resistor. Too small of a resistor will produce a V_{IL} level that is too high. Too large of a resistor will degrade the signal's rise time. A minimum value for the resistor is determined as follows:

$$\frac{(V_{CC}\ \text{max}) - (82062\ V_{IL}\ \text{max})}{(\text{TTL}\ I_{OL}\ \text{min.}) - (82062\ I_{IL}\ \text{max})} = \text{Resistor}$$

TASK REGISTER FILE

The Task Register File is a bank of registers used to hold parameter information pertaining to each command. These registers and their addresses are:

A2	A1	A0	READ	WRITE
0	0	0	(Bus Tri-Stated)	(Bus Tri-Stated)
0	0	1	Error Flags	Reduce Write Current
0	1	0	Sector Count	Sector Count
0	1	1	Sector Number	Sector Number
1	0	0	Cylinder Low	Cylinder Low
1	0	1	Cylinder High	Cylinder High
1	1	0	SDH	SDH
1	1	1	Status Register	Command Register

NOTE:
Registers are not cleared by $\overline{\text{RESET}}$

Error Register

This read-only register contains specific error status after the completion of a command. If any bit in this register is set, then the Error bit in the Status Register will also be set. The bits are defined as follows:

7	6	5	4	3	2	1	0
BBD	CRC	–	ID	–	AC	TK000	DM

Bit 7 - Bad Block Detect

This bit is set when an ID field has been encountered that contains a bad block mark. The bad block bit is set only during formatting. The 82062 will terminate a command if an attempt is made to read a sector that contains this bit.

Bit 6 - CRC Data Field

This bit is set when a data field CRC error has occurred. The sector buffer may still be read but will contain errors.

Bit 5 - Reserved.

Not used. Set to zero.

Bit 4 - ID Not Found

This bit is set when the desired cylinder, head, sector or size parameter cannot be found after 8 revolutions of the disk, or if an ID field CRC error has occurred.

Bit 3 - Reserved.

Not used. Set to zero.

Bit 2 - Aborted Command

This bit is set if a command was issued or in progress while DRDY (Pin 28) was deasserted or WR FAULT (Pin 30) was asserted. The Aborted Command bit will also be set if an undefined command is written into the COMMAND register, but an implied seek will be executed.

Bit 1 - TRACK 000

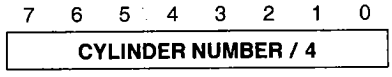
This bit is set only by the RESTORE command. It indicates that TRACK 000 (Pin 31) has not gone active after the issuance of 1024 stepping pulses.

Bit 0 - Data Address Mark

This bit is set during a READ SECTOR command if the Data Address Mark is not found after the proper Sector ID is read.

Reduce Write Current Register

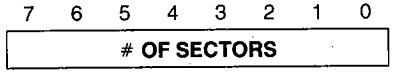
This register is used to define the cylinder number where RWC (Pin 33) is asserted:



The value (0–255) written into this register is internally multiplied by 4 to specify the actual cylinder where RWC is asserted. Thus a value of 01H will cause RWC to activate on cylinder 4, 02H on cylinder 8 and so on. RWC will be asserted when the present cylinder is greater than or equal to the cylinder indicated by this register. For example, one ST506 compatible drive requires precompensation on cylinder 128 (80H) and above. Therefore the REDUCE WRITE CURRENT register should be loaded with 32 (20H). A value of FFH will keep the RWC output inactive regardless of the actual cylinder number.

Sector Count Register

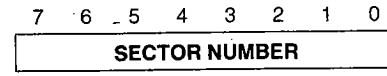
This register is used to define the number of sectors that need to be transferred to the buffer during a READ MULTIPLE SECTOR or WRITE MULTIPLE SECTOR command.



The value contained in the register is decremented after each sector is transferred to/from the sector buffer. A zero represents a 256 sector transfer, a one a 1 sector transfer, etc. This register is ignored when single sector commands are specified in the Command register.

Sector Number

This register holds the sector number of the desired sector:

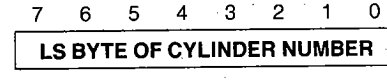


For a multiple sector command it specifies the first sector to be transferred. It is decremented after each sector is transferred to/from the sector buffer. The SECTOR NUMBER register may contain any value from 0 to 255. The ID Not Found bit will be set if the desired sector cannot be located on the track.

The SECTOR NUMBER register is also used to program the Gap 1 and Gap 3 lengths to be used when formatting a disk. See the WRITE FORMAT command description for further explanation.

Cylinder Number Low Register

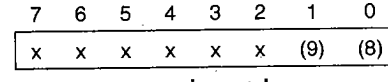
This register holds the lower byte of the desired cylinder number:



It is used in conjunction with the CYLINDER NUMBER HIGH register to specify a range of 0 to 1024 tracks.

Cylinder Number High Register

This register holds the two most significant bits of the desired cylinder number:



x = ignored

The 82062 contains a pair of registers that store the actual position where the R/W head are located. The CYLINDER NUMBER HIGH and LOW registers are considered the cylinder destination registers for seeks and other commands. The 82062 compares its internal registers to the destination registers and issues the number of steps in the right direction to make both sets of registers equal. After a command is executed, the internal cylinder position registers' contents are equal to the cylinder high/low registers. If a drive number change is detected on a new command, the 82062 automatically reads an ID field to update its internal cylinder position registers. This affects all commands except a RESTORE.

Sector/Drive/Head Register

The SDH register contains the desired sector size, drive number, and head number parameters. The format is shown below.

7	6	5	4	3	2	1	0
EXT	SECT SIZE		DRIVE		HEAD #		

Both head number and sector size are compared against the disk's ID field. Head select and drive select lines are not available as outputs from the 82062 and must be generated externally.

Bit 7, the extension bit (EXT), is used to extend the data field by seven bytes when using ECC codes for READ/WRITE SECTOR commands. When EXT = 1, the CRC is not appended to the end of the data field and the data field becomes "sector size + 7" bytes long. The CRC is checked on the ID field regardless of the state of EXT. The SDH byte written into the ID field is different than the SDH Register contents. The reccorded SDH byte does not have the drive number (DRIVE) written but does have the BAD BLOCK mark written. The EXT bit must not be set during the Format command.

Note that use of the extension bit requires the gap lengths to be modified as described in the WRITE FORMAT command description.

Status Register

The status register is a read-only register which informs the host of certain events. This register is a flow-through latch until the microprocessor reads it at which point the drive status lines are latched. The INTRQ line will be reset when this register is read. The format is:

7	6	5	4	3	2	1	0
BUSY	READY	WF	SC	DRQ	–	CIP	ERROR

Bit 7 - Busy

This bit is set whenever the 82062 is transferring data between its sector buffer and the disk and reflects the state of the $\overline{BCS}$ pin. When $\overline{BCS}$ is active, the host should not access the sector buffer or any 82062 register. The 82062 will be generating a $\overline{RD}$ or $\overline{WR}$ pulse every 1.6 μsec and the host must not interfere with these data transfers. Busy is cleared when the data transfer operation is completed.

During other non-data transfer commands, Busy should be ignored as it will go active for short periods.

Bit 6 - Ready

This bit reflects the state of the DRDY (Pin 28) line at the time the microprocessor reads the status register. Transitions on the DRDY line will abort a command and set the aborted command bit in the error register.

Bit 5 - Write Fault

This bit reflects the state of the WR FAULT (Pin 30) line. Transitions on this line will abort a command and set the aborted command bit in the error register.

Short transitions on DRDY and WR FAULT may not show up in the status register. These pins are not latched until the microprocessor reads the status and by that time the error condition may have disappeared. However the aborted command bit will be set to notify the host of an error. To hold short transitions on these pins it is recommended that they be latched.

Bit 4 - Seek Complete

This bit reflects the state of the SC (Pin 32) line. Commands which initiate a seek will pause until Seek Complete is set.

Bit 3 - Data Request

The Data request bit (DRQ) reflects the state of the BDRQ (Pin 36) line. It is set when the sector buffer should be loaded with data or read by the host processor, depending upon the command. The DRQ bit and the BDRQ line remain high until BRDY is sampled, indicating the operation has completed.

Bit 2 - Reserved

Not used. Set to zero.

Bit 1 - Command in Progress

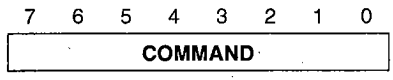
When this bit is set, a command is being executed and a new command should not be loaded until it is cleared. Although a command may be executing, the sector buffer is still available for access by the host processor. If CIP is set, only the status register can be read regardless of which register is selected.

Bit 0 - Error

This bit is an OR of the contents of the error register. Any bit being set in the error register sets this bit. This bit is cleared when a new command is loaded.

Command Register

This write-only register is loaded with the desired command:



The 82062 begins to execute immediately upon loading any value into this register. This register should not be written while the Busy or Command in Progress bits are set in the STATUS register. The INTRQ line (Pin 3) if set, will be cleared by a write to the COMMAND register.

Instruction Set

The 82062 WDC instruction set contains six commands. Prior to loading the command register, the host processor must first set up the Task Register File with the information needed for the command. Except for the COMMAND register, the registers may be loaded in any order. If a command is in progress, a subsequent write to the COMMAND register will be ignored. A command is finished when the command in progress (CIP) bit in the STATUS register is cleared. See the Command Section for an explanation of each command.

COMMAND	7	6	5	4	3	2	1	
RESTORE	0	0	0	1	R3	R2	R1	R0
SEEK	0	1	1	1	R3	R2	R1	R0
READ SECTOR	0	0	1	0	I	M	0	T
WRITE SECTOR	0	0	1	1	0	M	0	T
SCAN ID	0	1	0	0	0	0	0	T
WRITE FORMAT	0	1	0	1	0	0	0	0

R 3–0 = Rate Field

For 5 MHz WR Clock:

0000 — ≈35 μs
0001 — 0.5 ms
0010 — 1.0 ms
0011 — 1.5 ms
0100 — 2.0 ms
0101 — 2.5 ms
0110 — 3.0 ms
0111 — 3.5 ms
1000 — 4.0 ms
1001 — 4.5 ms
1010 — 5.0 ms
1011 — 5.5 ms
1100 — 6.0 ms
1101 — 6.5 ms
1110 — 7.0 ms
1111 — 7.5 ms

COMMAND	7	6	5	4	3	2	1
T =	Retry Enable						
T = 0	Enable Retries						
T = 1	Disable Retries						
M =	Multiple Sector Flag						
M = 0	Transfer 1 Sector						
M = 1	Transfer Multiple Sectors						
I =	Interrupt Enable						
I = 0	Interrupt at BDRQ time						
I = 1	Interrupt at end of command						

Programming the 82062

This section consists of two parts. The first part gives an explanation of each command, a flowchart showing the 82062's sequence of events, and the commands' sequence of events as seen by the host microprocessor. The second section shows flowcharts of general software routines and their PLM equivalent, for both polled and interrupt driven software.

The designer must remember that the 82062 expects a full sector buffer that can be isolated from the host during data transfers between the 82062 and the disk. Since the 82062 assumes a full sector buffer is available, it does not check for data overrun or underrun error conditions. If such a condition occurs, corruption of data will happen and the host will have no indication of an error. The design must guarantee against over-run and under-run conditions when not using the sector buffer approach.

Commands

A command is placed into the command register only after the Task Registers have been written with proper values. The Task Registers may be loaded in any order. A command, once started, can only be terminated by a hardware reset to the 82062. This may corrupt data on the disk by removing necessary control signals out of sequence.

The general sequence of a command is as follows:

— The host loads the Task Registers
— The host loads the Command Register
— The 82062 locates the correct cylinder
— Data transfer takes place
— The 82062 issues an interrupt

Restore Command – 0 0 0 1 R3 R2 R1 R0

The Restore command is used to position the heads to cylinder 0. This command must be issued to the 82062 on power-up to initialize internal registers. The user specified rate field (R3–R0) is stored internally for FUTURE use in commands with implied seeks. The step rate value is not used with this command. The actual stepping rate used is dependent upon the handshake delay between the 82062 issuing a step pulse and the drive returning a seek complete for each track (roughly 20 ms). After each step pulse is issued, the 82062 waits for a rising edge on the Seek Complete (SC) line before issuing the next pulse. If 8 index pulses are received without a rising edge on SC, the 82062 will switch to sampling the level of the SC line. If after 1,024 step pulses the Track 00 signal has not gone active, the 82062 will terminate the command and set the TRACK 000 bit in the Error Register. The command will terminate if WR Fault goes active or DRDY goes inactive at any time. Figure 13 is a flow chart of the command.

This command should precede the format command. The format command will be aborted if an ID field is not present (because the disk was never formatted) and

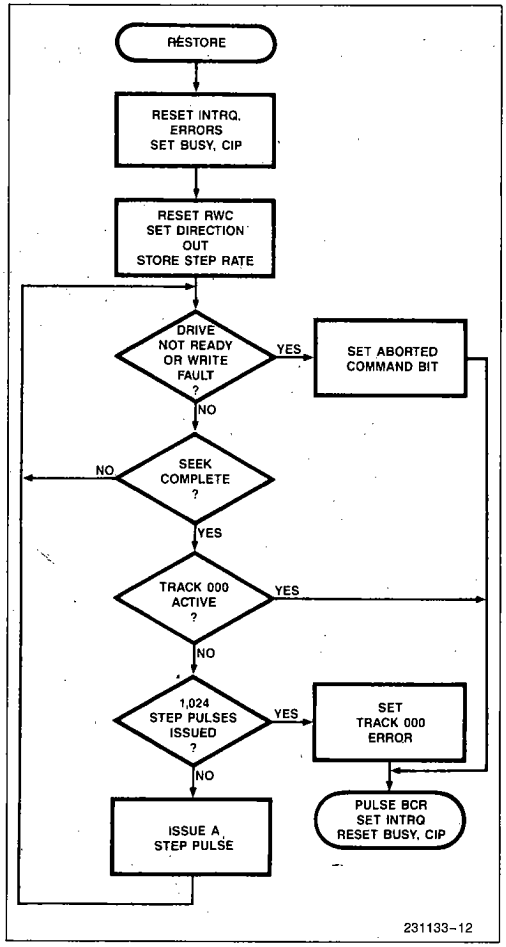


Figure 13. Restore Command Flow

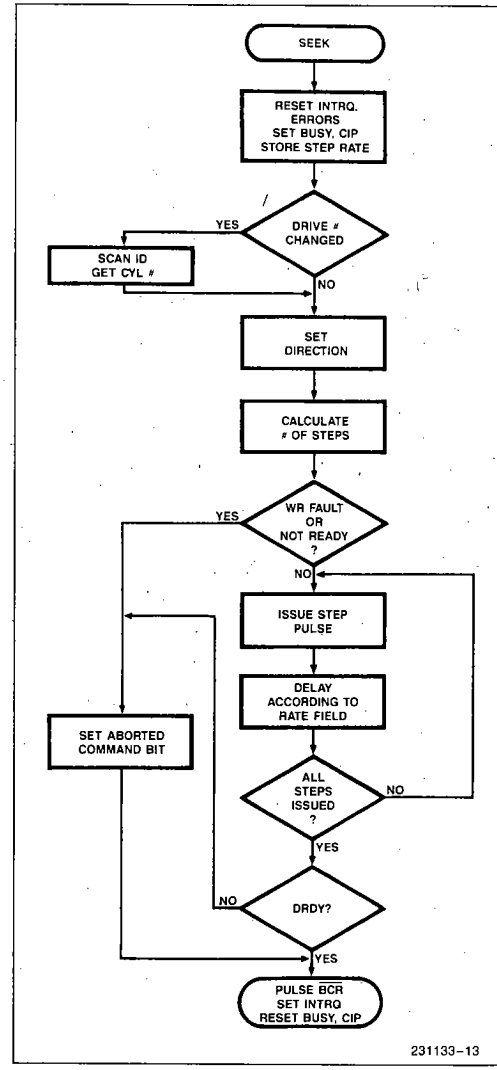


Figure 14. Seek Command Flow

a new drive is selected. Recall the 82062 will do a Scan ID to update internal registers when the drive is changed. This information is used to calculate the number of steps required to get to the destination cylinder. When the heads are positioned to track zero the 82062 will not try to read an ID field, but will issue the correct number of steps.

Seek Command – 0 1 1 1 R3 R2 R1 R0

The Seek command positions the heads to the cylinder specified in the Task Registers. The direction and number of step pulses issued is calculated by comparing the cylinder high/low registers to an internal "present position" cylinder register. The present position register is updated after all step pulses are issued and the command is terminated. The Seek Complete input is not checked.

The actual stepping rate is taken from the rate field bits (R3–R0) and stored for future use. The command terminates at once if WR FAULT goes active or DRDY goes inactive at any time. Figure 14 is a flowchart of the command.

Since the data transfer commands feature implied seeks, this command is of use mainly to those using multiple drives and software that can take advantage of overlapped seeks.

Scan ID Command – 0 1 0 0 0 0 0 T

The Scan ID command is used by both the 82062 and the host to update the SDH, the Sector Number, Cylinder and internal present position registers. Once the command is issued, the Seek Complete line is sampled until valid. The first ID field found, as indicated by the address mark, is loaded into the previously mentioned registers. The Bad Block bit will be set if detected, and the command will terminate. ID CRC errors will start the search sequence over for a maximum of 10 index pulses, but the registers will be loaded with whatever data the 82062 had perceived as ID information. Improper states on WR Fault on DRDY will terminate the command. Figure 15 is the flow chart of the command.

The main use for this command is to determine where the heads are currently located and what size the sectors are (i.e. 256, 512 etc.). Without this command, it would be necessary to recall the heads to track zero and then step out to the desired cylinder each time a drive was changed. Specifying the wrong sector size would yield an ID not found error. This command enables the system to read the disk drive to determine what size sectors were recorded.

Read Sector Command – 0 0 1 0 I M 0 T

The READ SECTOR command is used to transfer one or more sectors of data from the disk to the sector buffer. Upon receipt of the READ SECTOR command, the 82062 checks the CYLINDER NUMBER LOW/HIGH register pair against an internal cylinder position register to see if they are equal. If not, the direction and number of steps are calculated and a seek takes place. If an implied seek is performed, the 82062

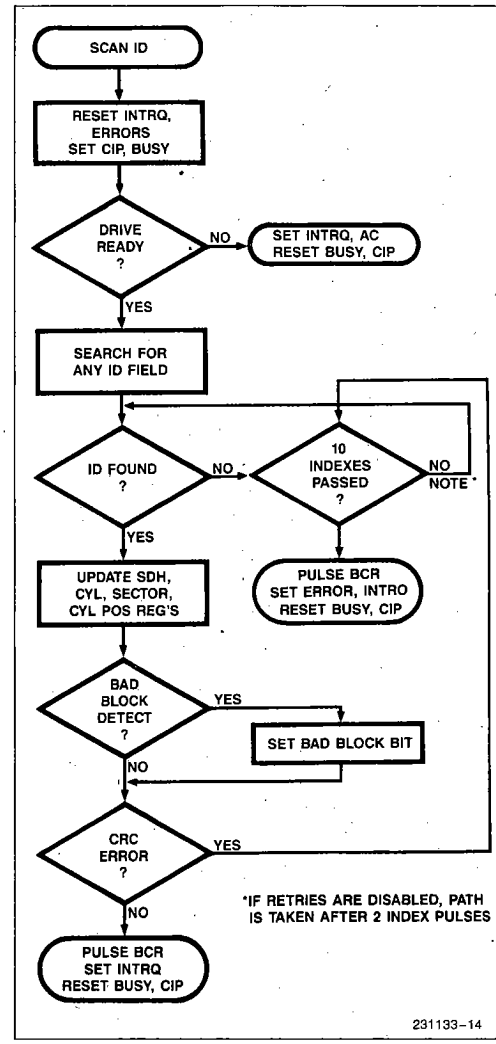


Figure 15. Scan ID Command Flow

will search until a rising edge of SC is received. The WR FAULT and DRDY lines are monitored throughout the command.

Once the Seek Complete (SC) line is high (with or without an implied seek having occurred), the search for an ID field begins. If T = 0 (retries enabled), the 82062 must find an ID with the correct cylinder number, head, sector size, and CRC within 10 revolutions, or a Scan ID and re-Seek will be performed. The search for the proper ID will again be tried for up to 10 revolutions. If the correct sector is still not found, the appropriate error bits will be set and the command terminated. Data CRC errors will also be retried for up to 10 revolutions (if T = 0).

If T = 1 (retries disabled), the ID search must find the correct sector within 2 revolutions or the appropriate error bits will be set and the command terminated.

Both the READ SECTOR and WRITE SECTOR commands feature a "simulated completion" to ease programming. DRQ/BDRQ will be generated upon detecting an error condition. This allows the same program flow for successful or unsuccessful completion of a command.

When the data address mark is found, the 82062 is ready to tranfer data to the sector buffer. After the data has been transferred, the I bit is checked. If I = 0, INTRQ is made active coincident with BDRQ, indicating that a transfer of data from the buffer to the host processor is required. If I = 1, INTRQ will occur at the end of the command, i.e. after the buffer is unloaded by the host.

The M bit is set for multiple sector transfers. When M = 0, one sector is transferred and the SECTOR COUNT register is ignored. When M = 1, multiple sectors are transferred. After each sector is transferred, the 82062 decrements the SECTOR COUNT register and increments the SECTOR NUMBER register. The next logical sector will be transferred regardless of any interleave. Sectors are numbered at format time.

Multiple sector transfers continue until the SECTOR COUNT register equals zero, or the BRDY line goes active (low to high). If the SECTOR COUNT register is non-zero (indicating more sectors are to be transferred but the buffer is full), BDRQ will be made active and the host must unload the buffer. After this occurs, the buffer will again be free to accept the remaining sectors from the 82062. This scheme enables the user to transfer more sectors than the buffer memory has capacity for.

In summary then, READ SECTOR operation is as follows:

When M = 0 (READ SECTOR)

(1)	Host:	Sets up parameters; issues READ SECTOR command.
(2)	82062:	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 0.
(3)	82062:	Finds sector specified; transfers data to buffer.
(4)	82062:	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 1.
(5)	82062:	Sets BDRQ = 1; DRQ = 1.
(6)	82062:	If I bit = 1 go to (9).
(7)	Host:	Reads contents of sector buffer.
(8)	82062:	Waits for BRDY, then sets INTRQ = 1: END.
(9)	82062:	Sets INTRQ = 1.
(10)	Host	Reads out contents of buffer; END.
(11)	82062:	If I = 1 wait for BRDY, then clear BDRQ; END.

When M = 1 (READ MULTIPLE SECTOR)

(1)	Host:	Sets up parameters; issues READ SECTOR command.
(2)	82062:	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 0.
(3)	82062:	Finds sector specified; transfers data to buffer.
(4)	82062:	Decrements SECTOR COUNT register; increments SECTOR NUMBER register.
(5)	82062:	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 0.
(6)	82062:	Sets BDRQ = 1; DRQ = 1.
(7)	Host:	Reads out contents of buffer.
(8)	82062:	Waits for BRDY.
(9)	82062:	When BRDY = 1, if Sector Count = 0 then go to (11).
(10)	82062:	Go to (2).
(11)	82062:	Set INTRQ = 1; End.

A flowchart of the READ SECTOR command is shown in Figures 16A and 16B.

Write Sector Command – 0 1 1 1 0 M 0 T

The WRITE SECTOR command is used to write one or more sectors of data to the disk from the sector buffer. Upon receipt of a WRITE SECTOR command the 82062 checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation is performed and a seek takes place. The WR FAULT and DRDY lines are checked throughout the command.

When the Seek Complete (SC) line is found to be true (with or without an implied seek having occurred), the BDRQ signal is made active and the host proceeds to load the buffer. Once BRDY goes high, the ID field with the specified cylinder number, head, and sector size is searched for. Once found, WR GATE is made active and the data is written to the disk. If retries are enabled (T = 0), and if the ID field cannot be found within 10 revolutions, a Scan ID and re-Seek are performed. If the correct ID field is not found within 10 additional revolutions, the ID Not Found error bit is set and the command is terminated. If retries are disabled, (T = 1) and if the ID field cannot be found within 2 revolutions, the ID Not Found error bit is set and the command is terminated.

During a WRITE MULTIPLE SECTOR command (M = 1), the SECTOR NUMBER register is decremented and the SECTOR COUNT register is incremented after the transfer to the disk takes place. During multiple sector transfers if BRDY is asserted after the first sector is transferred from the buffer, the 82062 will transfer the next sector before issuing BDRQ. The 82062 will set BDRQ and wait for the host processor to place more data in the buffer.

In summary then, the WRITE SECTOR operation is as follows:

When M = 0, 1 (WRITE SECTOR)

(1)	Host:	Sets up parameters; issues WRITE SECTOR command.
(2)	82062:	Sets BDRQ = 1, DRQ = 1.
(3)	Host:	Loads sector buffer with data.
(4)	82062:	Waits for BRDY = 0 to 1.
(5)	82062:	Finds specified ID field; writes sector to disk.
(6)	82062:	If M = 0, then set INTRQ = 1; END.
(7)	82062:	Increment SECTOR NUMBER register; decrement SECTOR COUNT register.
(8)	82062:	If SECTOR = 0, then set INTRQ = 1; END.
(9)	82062:	Go to (2).

A flowchart of the WRITE SECTOR command is shown in Figure 17.

Write Format Command 0 1 0 1 0 0 0 0

The WRITE FORMAT command is used to format one track using the Task Register File and the sector buffer. During execution of this command, the sector buffer is used for additional parameter information instead of sector data. Shown in Figure 18 is the contents of the sector buffer for a 32 sector/track format with an interleave factor of two. Each sector requires a two byte sequence. The first byte designates whether a bad block mark is to be recorded in the sector's ID field. A 00 Hex is normal; an 80H indicates a bad block mark for the sector. In the example of Figure 18, sector 04 will get a back block mark recorded. Any attempt to access sector 4 in the future will terminate the command.

The second byte indicates the logical sector number to be recorded. This allows sectors to be recorded with any interleave factor desired. The remaining memory in the sector buffer may contain any value. Its only purpose is to generate a BRDY to tell the 82062 to begin formatting the track. An implied seek is in effect on this command. As for other commands, if the drive number has been changed an ID field will be scanned for cylinder position information before the implied seek is performed. If no ID field can be read (because the track had been erased or because an incomplete format had been used), an ID Not Found error will result and the WRITE FORMAT command will be aborted. This can be avoided by issuing a RESTORE command before formatting.

The SECTOR COUNT register is used to hold the total number of sectors to be formatted (01H = 1 sector; 00H = 256 sectors), while the SECTOR NUMBER register holds the number of bytes (minus three) to be used for Gap 1 and Gap 3. For instance, if the SECTOR COUNT register value is 02H and the SECTOR NUMBER register value is 00H, then 2 sectors are written on a track and 3 bytes of 4EH are written for Gap 1 and Gap 3. The data fields are filled with FFH and the CRC is automatically generated and appended. All gaps are filled with 4EH. After the last sector is written, the track is filled with 4EH until the index pulse terminates the write. The Gap 3 value is deter-

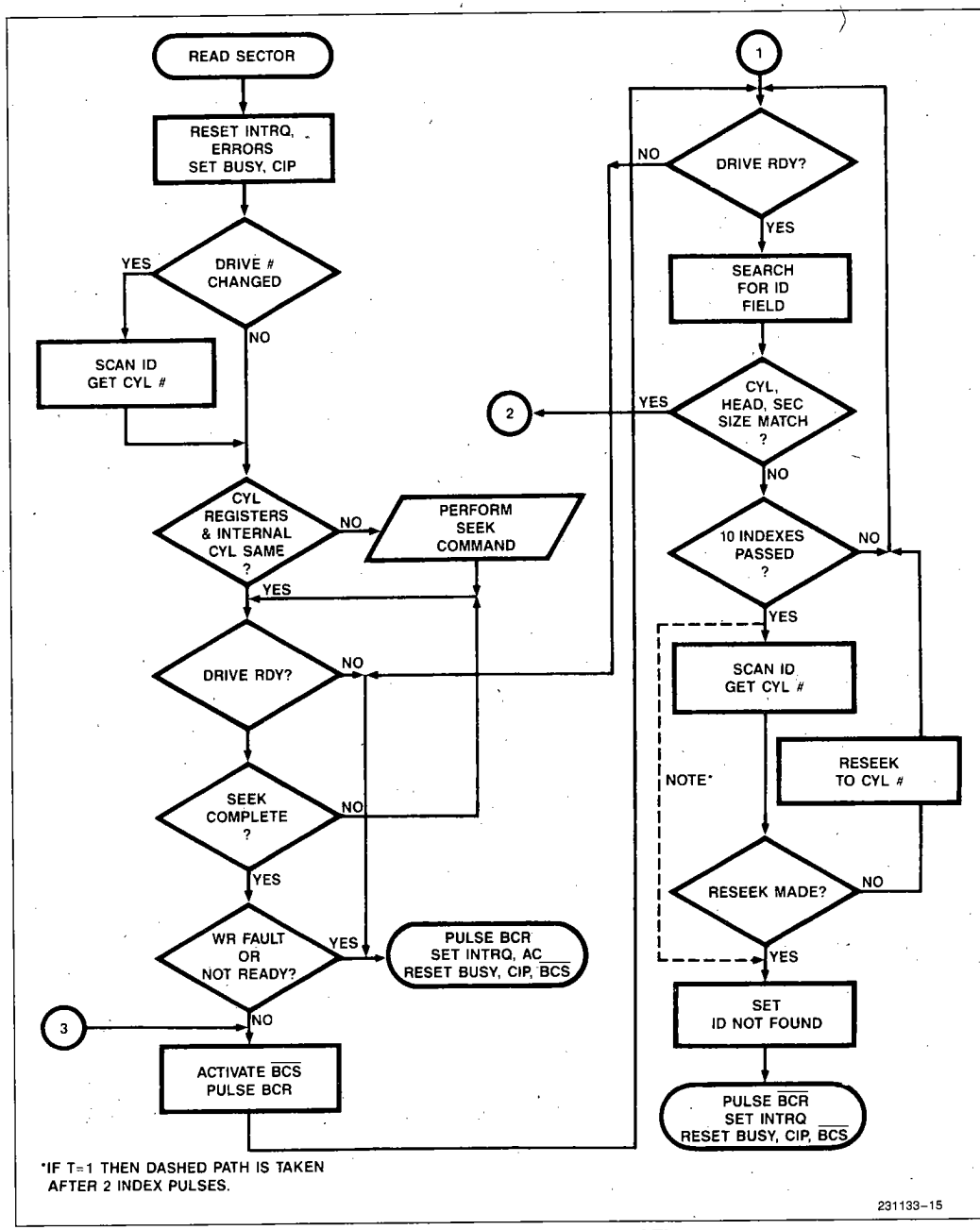


Figure 16A. Read Sector Command Flow

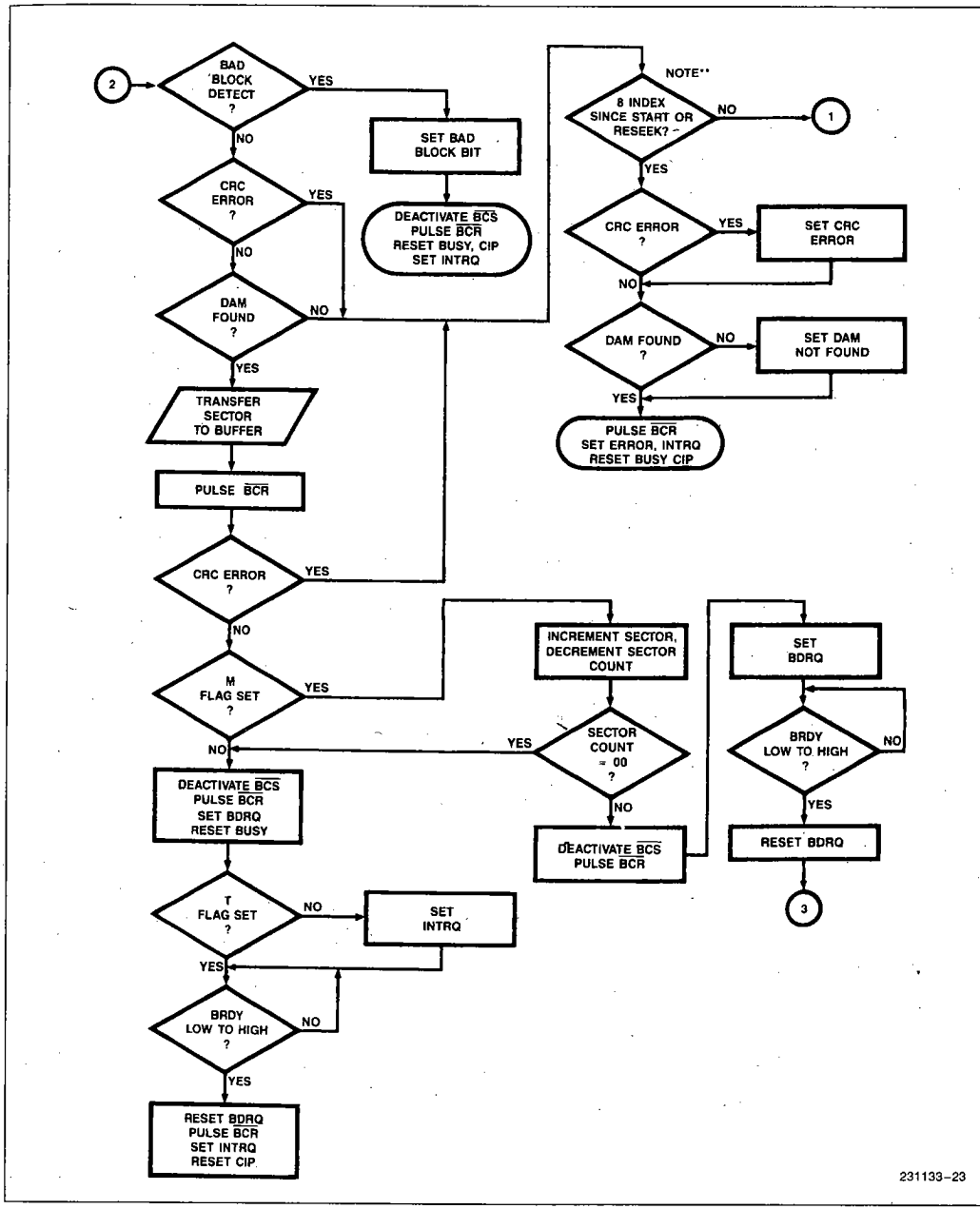


Figure 16B. Read Sector Command Flow

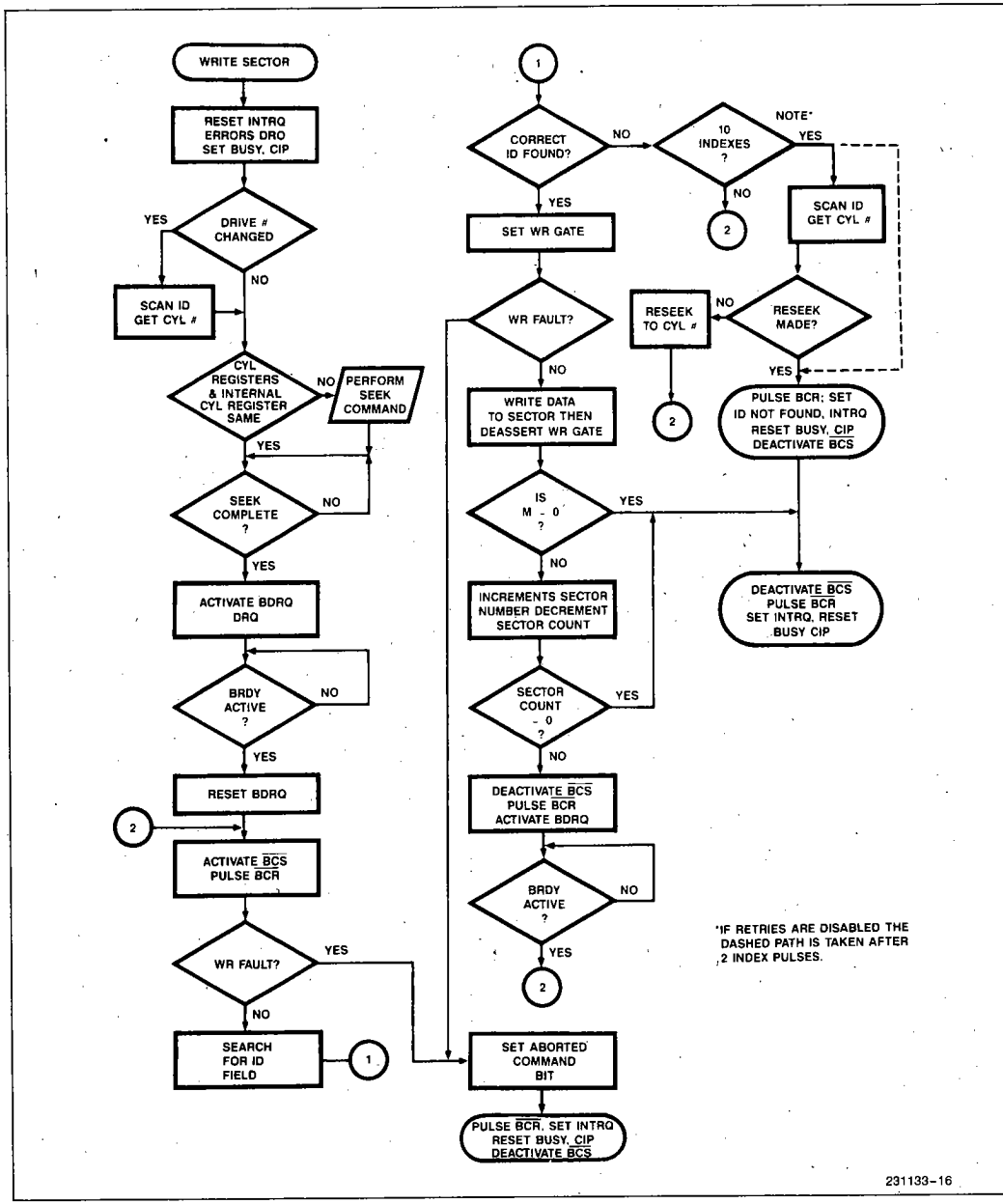


Figure 17. Write Sector Command Flow

00	00	00	10	00	01	00	11	00	02	00	12	00	03	00	13
80	04	00	14	00	05	00	15	00	06	00	15	00	07	00	17
00	08	00	18	00	09	00	19	00	0A	00	19	00	0B	00	1B
00	0C	00	1C	00	0D	00	1D	00	0E	00	1E	00	0F	00	1F
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 18. Sector Buffer Contents For Format

mined by the drive motor speed variation, data sector length, and the interleave factor. The interleave factor is only important when 1:1 (no) interleave is used. The formula for determining the minimum Gap 3 length value is:

Gap 3 = (2 * M * S) + K + E

M = motor speed variation (e.g., 0.03 for ±3%)

S = sector length in bytes

K = 25 for interleave factor of 1

K = 0 for any other interleave factor

E = 7 if the sector is to be extended

As with all commands, a WR FAULT or drive not ready condition, will terminate execution of the WRITE FORMAT command. Figure 19 shows the format that the 82062 will write on the disk. The extend bit in the SDH register must not be set during the Format command.

A flowchart of the WRITE FORMAT command is shown in Figure 20.

SOFTWARE SECTION: GENERAL PROGRAMMING

This section describes the software needed to communicate with the 82062 in order to store and retrieve data. This chapter describes the software in a general manner and Appendix B contains the actual implementation used to exercise the 82062 SBX board.

Polled Mode

As discussed in the Polled Interface Section, the 82062 does not directly support polled operation for data transfers without the addition of hardware. This section is based upon the polled interface as described in the Polled Interface Section.

The six 82062 commands can be divided into two groups, those with data transfers and those without. The commands that do not use the sector buffer are: Restore, Seek and Scan ID. The functions of each command are explained in the Commands Section. Figure 21 is a flowchart of a polled operation and a PLM example.

The last status that was read will contain any error conditions that might have occurred during the command.

For commands that do make use of the sector buffer, the size of the sector buffer will affect the software. If the sector buffer is equal in size to one sector, then a carry out of an address counter (for the sector buffer) as the buffer is being filled will indicate to the 82062 that the command should continue. If the sector buffer

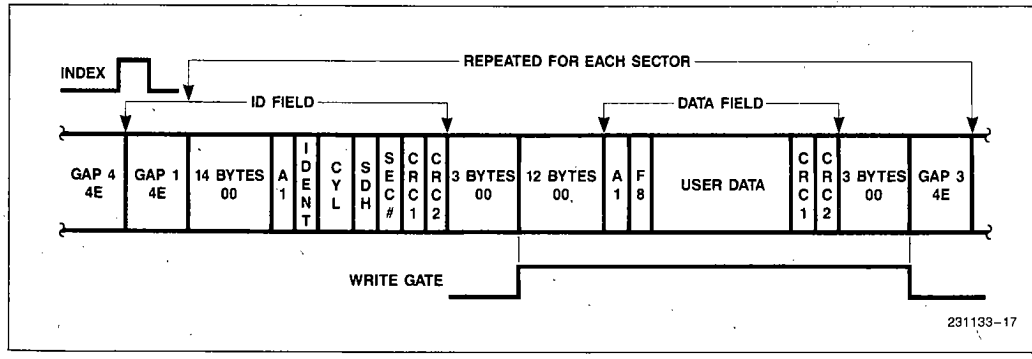


Figure 19. 82062 Sector Format

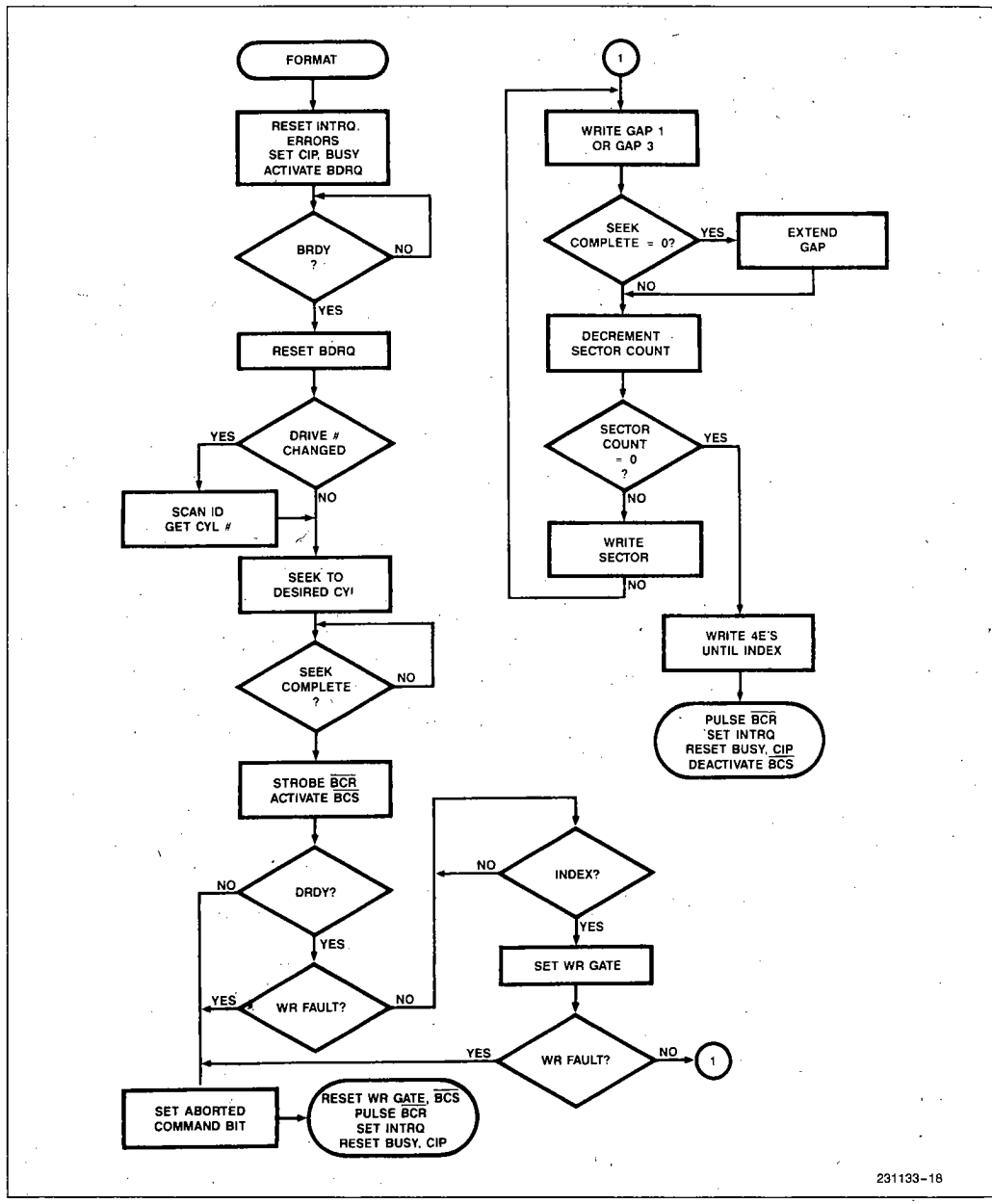


Figure 20. Write Format Command Flow

size is equal to two or more disk sectors, and only one sector is being transferred, then the carry out signal would not go active, and the 82062 will be forever waiting for BRDY. In this case an I/O port would have to be used to generate this signal for the 82062 so that command execution can finish. Figure 22 is a flowchart of the READ SECTOR command, and its PLM representation. The WRITE SECTOR and FORMAT TRACK commands are equivalent in terms of software interfacing. Their flowcharts and their PLM equivalents are shown in Figure 23.

Once the command register is written the 82062 requests a data transfer before locating the proper track. Once the buffer is filled and BRDY is asserted, the 82062 will locate the target track and sector. If the ID is not located before the selected number of retries have occurred, the 82062 will terminate the command. The data transferred to the sector buffer will not have been used. Once the command has finished (i.e., CIP = 0), the status and error registers will inform the host of an error.

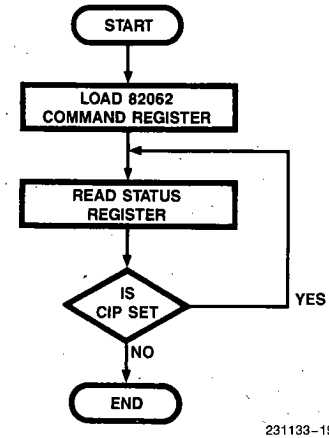
Figure 24 is the PLM routine that allows for all six of the commands. It differs from the READ and WRITE routines in that the direction that data is to be transferred is determined by the command.

Figure 24 also works for multiple sector transfers. However, the BRDY signal must be generated in hardware (the carry-out of an address counter).

Interrupt Mode

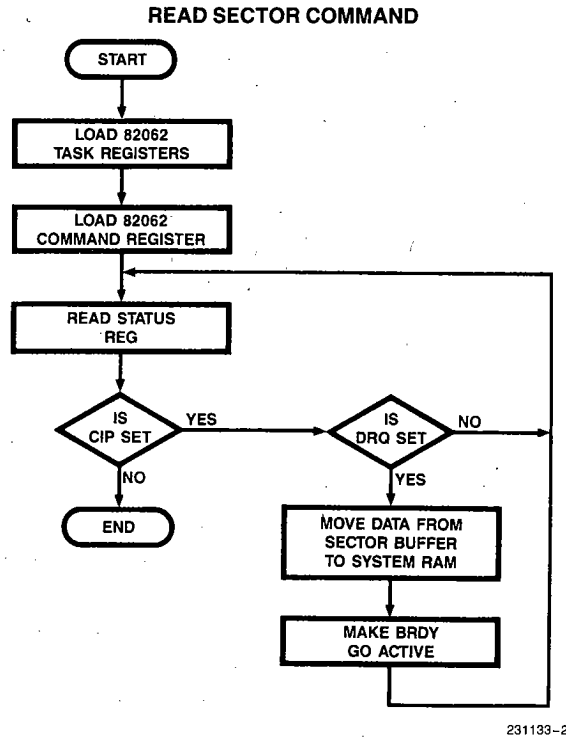
Interrupt driven software is chosen when the microprocessor must execute other tasks and cannot sit waiting for the disk to reposition its heads, as in a polled environment. The delay in repositioning heads can be anything from a couple of milliseconds to a second or more.

The 82062's interrupt (INTRQ) pin goes active to indicate that the command has finished. The READ SECTOR command provides the programmable choice of having the interrupt occur at the end of the data transfer or the normal end of the command. The reason for this option is that when the 82062 signals that a data transfer is required (via BDRQ, DRQ) the disk has been read and the data has been placed in the buffer. The host would remove the data and issue BRDY. The 82062 would then issue an interrupt indicating that the command has finished. The interrupt procedure would only have to read the status register. If the interrupt is issued at BDRQ the host would remove the buffer data



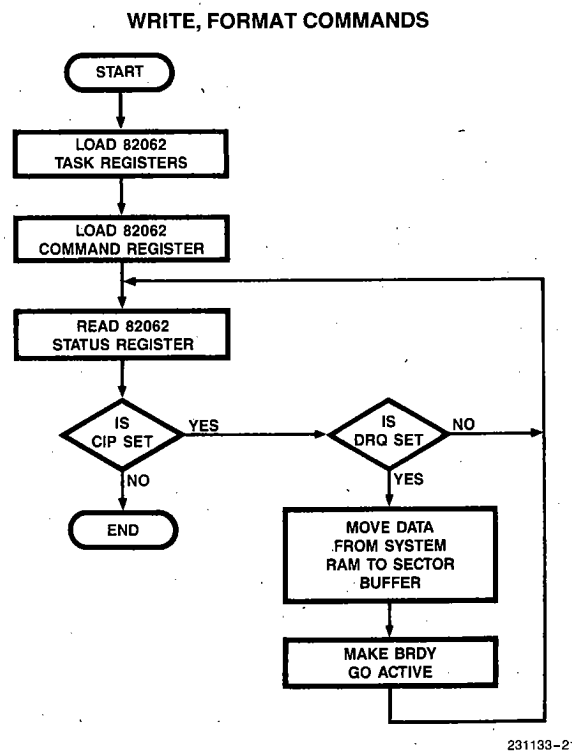

```
Disk$Operation: Procedure;
   Call Write$82062$Task$Reg's; /* Write Task Registers */
   Output (Command$Reg) = Command;
   Status = Input (Status$Reg); /* Read Status Reg */
   Do while Status and CIP = CIP; /* Wait until command finishes */
      Status = Input (Status$Reg);
   End;
End Disk$Operation;
```

Figure 21. Polling Status




```
Disk$Operation: Procedure;
    Call Write$82062$Task$Regs;
    Output (Command $ Reg) = Command;
    Status = Input (Status$Reg);
    Do while Status and CIP = CIP;
        If Status and DRQ = DRQ then Do;
          Call Read$Data$From$Buffer;
          Output (BRDY$PORT) = 01;
        End;
        Status = Input (Status$Port)
    End;
End Disk$Operation;
```

Figure 22. Polling For Read Data




```
Disk$Operation: Procedure;
  Call Write$82062$Task$Regs;
  Output (Command$Reg) = Command;
  Status = Input (Status$Reg);
  Do while status and CIP = CIP;
    If status and DRQ = DRW then do;
      Call Write$Data$to$Buffer;
      Output (BRDY$Port) = 01; /* Make BRDY go active */
    End;
    Status = Input (Status$Reg)
  End;
End Disk$Operation;
```

Figure 23. Polling For Write Data

```
Disk$Operation: Procedure;
   Call Write$82062$Task$Regs; /* Write registers */
   Output (Command$Reg) = Command; /* Start command */
   Status = Input (Status$Reg); /* Read status */
   Do while status and CIP = CIP; /* Is a command in progress */
     If status and DRQ = DRQ then do; /* Data transfer? = yes */
       If command = Read$Sector then
         Call Read$Data$From$Buffer; /* Remove data */
       Else Call Write$Data$to$Buffer; /* Send data */
       Output (BRDY$PORT) = 01; /* Toggle BRDY 0 to 1 */
     End;
End Disk$Operation;
```

Figure 24. Complete Polled Flow

```
Start$Disk$Operation: Procedure;
    Call Write$82062$Task$Reg's;
    Output (Command $ Reg) = Command;
End Start$Disk$Operation;
```

Figure 25. Interrupt Mode; Starting a Disk Transfer

and generate BRDY. At this point the status and error registers contain valid information. Generating an interrupt at BDRQ time may save some systems some software overhead.

The WRITE SECTOR and FORMAT commands do not have this option because the sector buffer is filled before the track and sector are located. Hence, there can be significant delays between asking for data and the command terminating.

In an interrupt driven environment, the 82062 can interface to a DMA controller for data transfers between the sector buffer and the host's RAM. If a DMA controller is not available an interrupt must be generated via the BDRQ line. However, BDRQ can stay active for long periods of time (until BRDY is generated). The interrupt sensing logic must take this into account to avoid being retriggered constantly. Intel's 8259A Interrupt Controller 8259A provides that capability. It should be programmed for edge triggered interrupts or the end of interrupt byte must not be issued until BDRQ is removed to prevent retriggering.

Figure 25 is a PLM example of starting a disk operation in an interrupt driven environment. The command starts, and some indefinite amount of time later an interrupt would be generated, indicating service is required.

If a DMA controller is used, it would have to be programmed and initialized before the command is issued to the 82062. Recall that once a data transfer between the microprocessor and 82062 has finished, BRDY must be set high. As long as BRDY is generated from hardware, no microprocessor intervention is needed. If BRDY is generated by an I/O port the microprocessor will have to perform this function (this will be the case with any system that has a sector buffer larger than one sector). (One option could be to generate an interrupt from the terminal count pin of the DMA controller. The microprocessor would then issue a BRDY.) Data transfers between host RAM and the sector buffer would be handled without microprocessor intervention. The interrupt would then signal that the command has finished as shown in Figure 26. The only operation the host processor would perform is to check the status register of the 82062 for any error conditions.

If BDRQ is used to generate an interrupt in addition to the normal interrupt, then the routines shown in Figure 27 will check the status register to see if a data transfer should be executed or if the command is finished. If DRQ is not set, the command has finished and any error conditions would be in the status register.

Another possibility would be to have separate interrupt routines for the two possible sources of interrupts

```
End$of$Transfer: Procedure Interrupt;
    Status = Input (Status$Register);
    Output (8259A PIC) = End$of$Interrupt;
End End$of$Transfer;
```

Figure 26. Checking Status via Interrupt

```
Service$Disk$Controller: Procedure Interrupt;
    Status = Input (Status$Port);
    If Status and DRQ = DRQ then
        Call Transfer$Data$To/From$Buffer;/* Enable DMAC */
    Output (8259A PIC) = End$of$Interrupt;
End Service$Disk$Controller;
```

Figure 27. Complete Interrupt Procedure

(INTRQ, BRDQ). There would then be no need to test the status to see which interrupt had occurred.

APPLICATION EXAMPLE

This section shows an application using the 82062 interfaced to the SBX bus. A quick overview of the SBX bus is provided (pin descriptions, general wave forms) as a background for the application. Designing the 82062 onto an SBX Multimodule board was chosen to highlight the size and complexity differences between earlier TTL, MSI, LSI-based disk controller boards and what is possible using the 82062. Both the hardware and software sections will be applicable to most other designs using the 82062. This design example is called SBX82062 and does not represent a real product offered by Intel Corporation. Appendix C contains the schematic of the SBX board.

The advantage of the SBX Multimodule is that it permits the system to be tailored for specific needs with a minimum of effort. The advantage of an SBX based disk controller is that a current system can make use of the capacity, reliability and speed of a hard disk with no (or minimal) hardware redesign.

iSBX Bus Multimodule Boards

The iSBX Multimodule boards are small, specialized, I/O mapped boards which plug onto base boards. The iSBX boards connect to the iSBX bus connector and convert the iSBX bus signals to a defined I/O interface.

Base Boards

The base board decodes I/O addresses and generates the chip selects for the iSBX Multimodule boards. In 8-bit systems, the base board decodes all but the lower three addresses in generating the iSBX Multimodule board chip selects. In 16-bit systems, the base board decodes all but the lower order four addresses in generating the iSBX Multimodule board chip selects. Thus, a base board would normally reserve two blocks of 8 I/O ports for each iSBX socket it provides.

There are two classes of base boards, those with Direct Memory Access (DMA) support and those without. Base boards with DMA support are boards with DMA controllers on them. These boards, in conjunction with an iSBX Multimodule board (with DMA capability), can perform direct I/O to memory or memory to I/O operations.

iSBX Bus Interface

The iSBX bus interface can be grouped into six functional classes:

1. Control Lines
2. Address and Chip Select Lines
3. Data Lines
4. Interrupt Lines
5. Option Lines
6. Power Lines

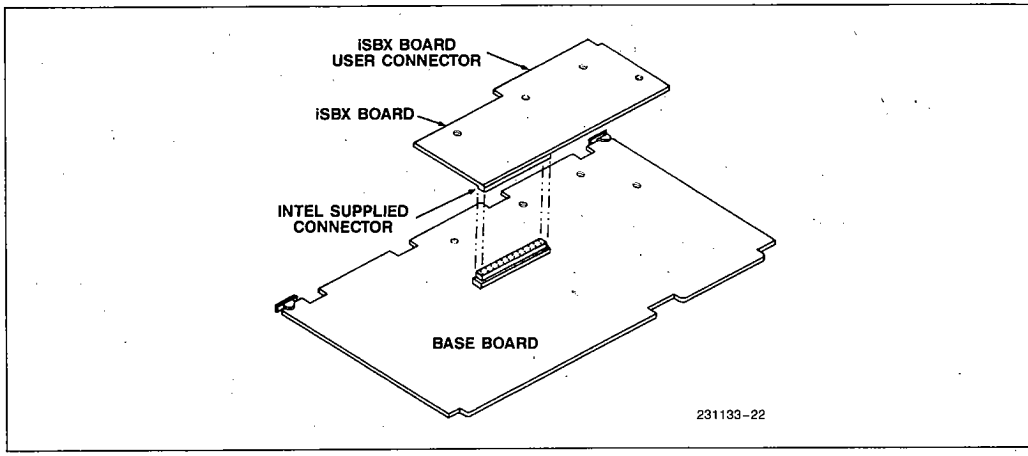


Figure 28. iSBX Multimodule Board Concept (Double Wide)

Control Lines

The following signals are classified as control lines:

COMMANDS:
- $\overline{\text{IORD}}$ (I/O Read)
- $\overline{\text{IOWRT}}$ (I/O Write)

DMA:
- $\overline{\text{MDRQT}}$ (DMA Request)
- $\overline{\text{MDACK}}$ (DMA Acknowledge)
- TDMA (Terminate DMA)

INITIALIZE:
- RESET

CLOCK:
- MCLK (iSBX Multimodule Clock)

SYSTEM CONTROL:
- $\overline{\text{MWAIT}}$
- $\overline{\text{MPST}}$ (iSBX Multimodule Board Present)

Command Lines ($\overline{\text{IORD}}$, $\overline{\text{IOWRT}}$)

The command lines are active low signals which provide the communication link between the base board and the iSBX Multimodule board. An active command line, conditioned by chip select, indicates to the iSBX Multimodule board that the address lines are valid and the iSBX Multimodule board should perform the specified operation.

DMA Lines (MDRQT, $\overline{\text{MDACK}}$, TDMA)

The DMA lines are the communication link between the DMA controller device on the base board and the iSBX Multimodule board. MDRQT is an active high output signal from the iSBX Multimodule board to the base board's DMA device requesting a DMA cycle. $\overline{\text{MDACK}}$ is an active low input signal to the iSBX Multimodule board from the base board DMA device acknowledging that the requested DMA cycle has been granted. TDMA is an active high output signal from the iSBX Multimodule board to the base board. TDMA is used by the iSBX Multimodule board to terminate DMA activity. The use of the DMA lines is optional as not all base boards will provide DMA channels and not all iSBX Multimodule boards will be capable of supporting a DMA channel.

Initialize Lines (Reset)

This input line to the iSBX Multimodule board is generated by the base board to put the iSBX Multimodule board into a known internal state.

Clock Lines (MCLK)

This input to the iSBX Multimodule board is a timing signal. The 10 MHz (+0%, −10%) frequency can vary from base board to base board. This clock is asynchronous from all other iSBX bus signals.

System Control Lines ($\overline{\text{MWAIT}}$, $\overline{\text{MPST}}$)

These output signals from the iSBX Multimodule board control the state of the system.

An active $\overline{\text{MWAIT}}$ (Active Low) will put the CPU on the board into wait states providing additional time for the iSBX Multimodule board to perform the requested operation. $\overline{\text{MWAIT}}$ must be generated from address

(address plus chip select) information only. If $\overline{\text{MWAIT}}$ is driven active due to a glitch on the CS line during address transitions, $\overline{\text{MWAIT}}$ must be driven inactive in less than 75 ns.

The iSBX Multimodule board present ($\overline{\text{MPST}}$) is an active low signal (tied to signal ground) that informs the base board I/O decode logic that an iSBX Multimodule board has been installed.

Address and Chip Select Lines

The address and chip select lines are made up of two groups of signals.

Adress Lines: MA0–MA2

Chip Select Lines: $\overline{\text{MCS0}}$–$\overline{\text{MCS1}}$

The base board decodes I/O addresses and generates the chip selects for the iSBX Multimodule boards. The base board decodes all but the lower order three addresses in generating the iSBX Multimodule board chip selects.

Address Lines (MA0–MA2)

These positive true input lines to the iSBX Multimodule boards are generally the least three significant bits of the I/O address. In conjunction with the command and chip select lines, they establish the I/O port address being accessed. In 16-bit systems, MA0–MA2 may be connected to ADR1–ADR3 of the base board address lines.

Chip Select Lines (MCS0–MCS1/)

In an 8-bit system, these input lines to the iSBX Multimodule board are the result of the base board I/O decode logic. $\overline{\text{MCS}}$ is an active low signal which conditions the I/O command signals and thus enables communication with the iSBX Multimodule boards.

The SBX82062 Design Example

The SBX82062 Multimodule board will interface an ST506 compatible drive to any host board having an SBX connector. Two restrictions on the disk drive are that there is a maximum of 1024 cylinders and/or 8 heads. The SBX connector cannot supply the power-up current requirements of the drive. The drive must be connected directly to the power supply. The SBX82062 in Appendix C does not support DMA transfers. The version in Appendix D does support DMA transfers. Since this multimodule has a 2 kbyte sector buffer, the host microprocessor must generate a BRDY by accessing an I/O port during data transfers.

The software for communicating to the SBX board is intended to be interrupt driven. Polling for data transfers is not supported. Reading the status without an interupt is not recommended. During the times the 82062 is accessing the sector buffer, the SBX82062 will isolate itself from the host. To support polling, a hardware generated busy pattern should be driven onto the hosts's data bus as is shown in the Polled Interface section. The sector buffer stores up to 2 kbytes of disk data, for multiple sector transfers. The SBX board only interfaces to one drive (for space reasons), but four drives could be used with the addition of a read data multiplexor (one IC) and the drive data cables.

Microprocessor Interface

Figure 29 is a block diagram of the SBX82062's microprocessor interface. The I/O port assignments are listed in Table 1. The functional blocks of the interface are:

Sector Buffer Isolation Logic

Wait State Logic

Sector Buffer

Sector/Drive/Head Register Logic

Table 6-1. I/O Port Assignments

Port Address	Read	Write
80H	Sector Buffer	Sector Buffer
82H	Error Reg	RWC Reg
84H	Sector Count	Sector Count
86H	Sector Number	Sector Number
88H	Cylinder Low	Cylinder Low
8AH	Cylinder High	Cylinder High
8CH	SDH Reg	SDH Reg
8EH	Status Reg	Command Reg
90H	None	None
92H	None	Asserts $\overline{\text{BCR}}$
94H	None	Asserts BRDY

NOTE:
Address assignments are determined by the host board.

Sector Buffer Isolation Logic

The host will be isolated from the SBX board whenever the 82062 is accessing its sector buffer which is enabled by $\overline{\text{BCS}}$. The host's control signals, $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{MCSO}}$, and $\overline{\text{MCS1}}$ and data bus are also disabled at the same time to prevent any data in the sector buffer from being corrupted. The host should wait for an interrupt before reading the 82062's Status register. Attempting to read the SBX board while $\overline{\text{BCS}}$ is active will return invalid data, since the SBX board will have the data bus tri-stated.

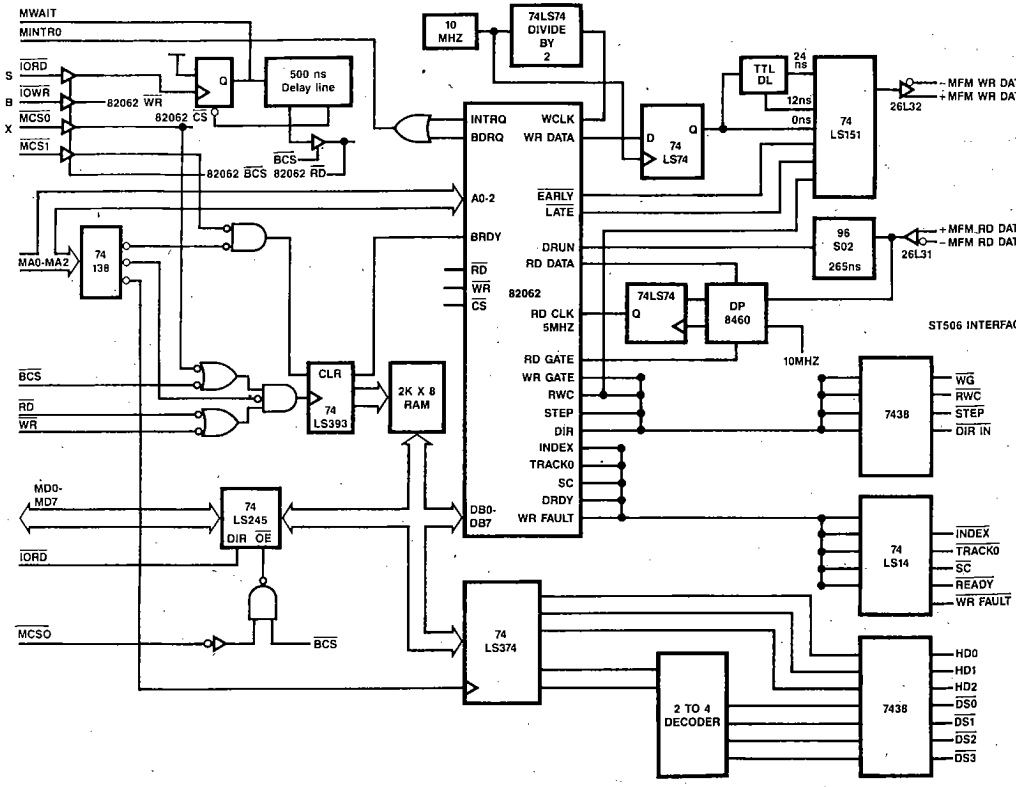

MWAIT
MINTR0
10 MHZ
74LS74 DIVIDE BY 2
S
B
X
IORD
IOWR
MCS0
MCS1
82062 WR
82062 CS
Q
500 ns Delay line
82062 BCS
82062 RD
BCS
INTRQ
BDRQ
WCLK
WR DATA
A0-2
BRDY
RD
WR
CS
82062
EARLY
LATE
DRUN
RD DATA
RD CLK 5MHZ
RD GATE
WR GATE
RWC
STEP
DIR
INDEX
TRACK0
SC
DRDY
WR FAULT
DB0-DB7
D
Q
74 LS74
TTL DL
24 ns
12ns
0ns
74 LS151
- MFM WR DATA
+ MFM WR DATA
26L32
96 S02 265ns
+ MFM RD DATA
- MFM RD DATA
26L31
74LS74
DP 8460
ST506 INTERFACE
10MHZ
7438
WG
RWC
STEP
DIR IN
74 LS14
INDEX
TRACK0
SC
READY
WR FAULT
MA0-MA2
74 138
BCS
RD
WR
CLR
74 LS393
2K X 8 RAM
MD0-MD7
74 LS245
DIR OE
IORD
MCS0
BCS
74 LS374
2 TO 4 DECODER
7438
HD0
HD1
HD2
DS0
DS1
DS2
DS3

Figure 29

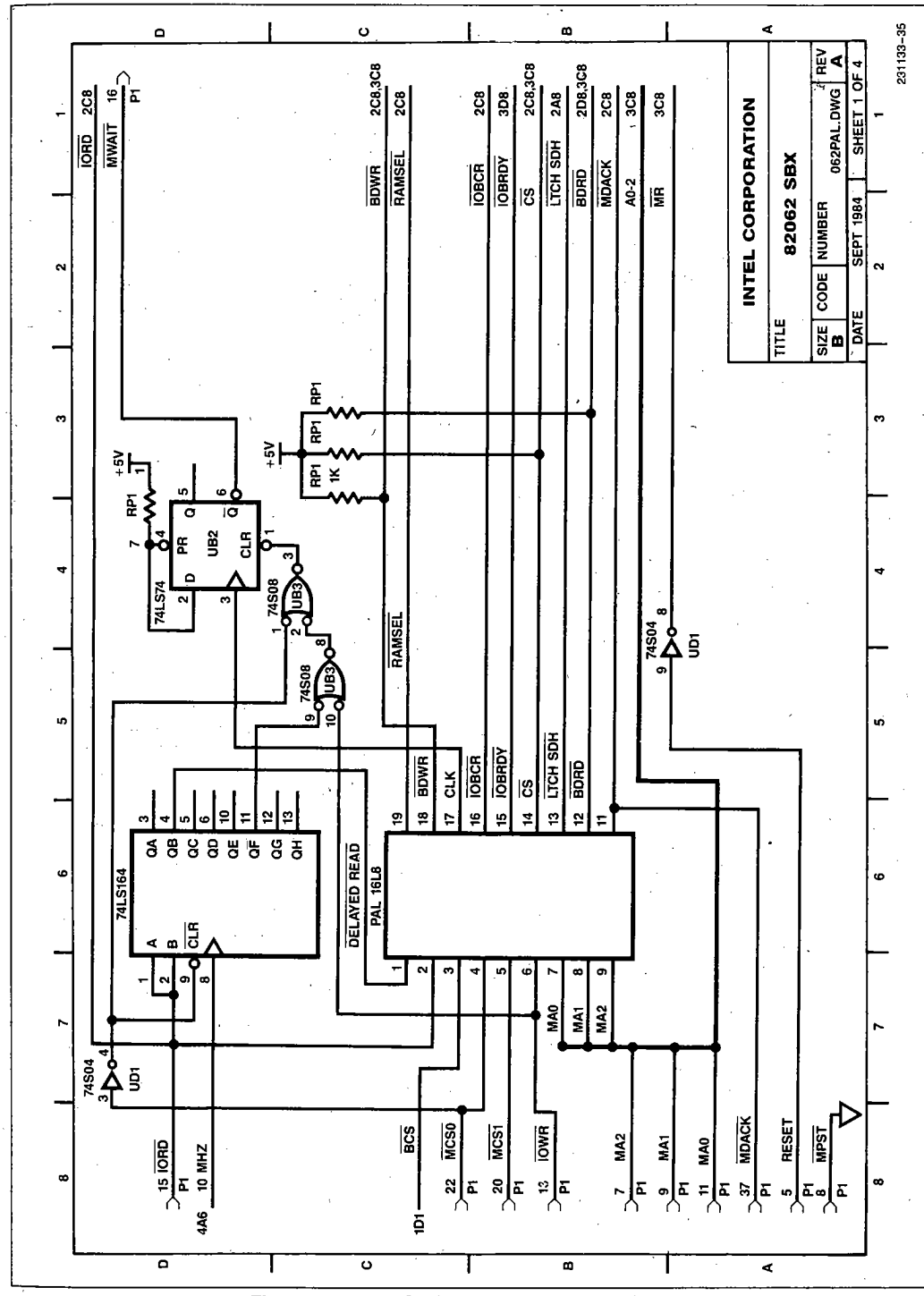


Figure 29. 82062 SBX Multimodule Block Diagram

231133-002

Wait State Logic

The wait state logic drives the 'not ready' line, $\overline{MWAIT}$, active whenever the host reads the SBX board. $\overline{MWAIT}$ does not go active for buffer or 82062 register writes. This logic was required for two reasons. First, a delayed read is generated, because the address setup to $\overline{RD}$ margin of the SBX bus is less than the 82062's needs (50 ns vs 100 ns). Second, the $\overline{RD}$ to data valid access period of the 82062 (375 ns), is greater than the SBX bus' full speed read cycle (275 ns) permits. $\overline{MWAIT}$ is deactivated after allowing for the delayed $\overline{RD}$ and the access period of the 82062. This delay is accomplished with a 500 ns delay line. The first tap at 100 ns generates the read request to allow for the address setup margin. The next tap 400 ns later removes $\overline{MWAIT}$ to allow the host to continue.

Sector Buffer

The sector buffer consists of an address counter (using '1s393's) and a 2 kbyte static RAM. The address counter is incremented on the trailing edge of a valid $\overline{RD}$ or $\overline{WR}$ cycle, either host microprocessor or 82062 initiated. The counter is reset by a hardware reset, the 82062 buffer reset $\overline{BCR}$, or by accessing an I/O port to provide software control. The 82062 will issue $\overline{BCR}$ each time $\overline{BCS}$ changes state (i.e. twice per sector). Resetting the buffer counter can be put under software control for multiple sector transfers. BRDY going high tells the 82062 that the buffer is available for its use. BRDY is generated by the address counter, by filling or emptying the entire buffer in multiple sector transfers, or from an I/O port when single sector transfers are done (since single sectors won't use all 2 kbytes of the buffer, the hardware signal will not be generated). When the 82062 is using the buffer, $\overline{BCS}$ will be low, and the $\overline{RD}$ or $\overline{WR}$ line will be pulsed every 1.6 microseconds.

When the 82062 is using the buffer it prevents access by the host by tristating the read, write, select and data lines with a low on $\overline{BCS}$.

SDH Register Logic

The drive and head select bits must be latched externally to the 82062, since these outputs are not provided. An 8 bit latch is strobed on the trailing edge of the $\overline{WR}$ pulse when the SDH register is selected. The two drive select bits are then demultiplexed to provide a one of four drive select line. If multiple drives are used then these outputs would also be used to select which disk's read data line would be gated into the PLL.

Interrupts

While the interrupt line is programmable (to notify of an end of command or data transfer request for the Read Sector command only), software will ensure that the interrupt from the 82062 signifies command termination. The BDRQ line is OR'ed with the 82062's INTRQ line or BDRQ can generate its own interrupt. BDRQ is also gated off-board for a DMA controller.

Disk Interface

Figure 30 is a block diagram of the interface between the 82062 and the disk drive. The functional blocks are:

Write Data Logic

Read Data Logic (PLL)

Drive Control

Write Data Logic

The WR DATA output requires a D flip-flop clocked at 10 MHz to complete the conversion of data to MFM. The output of this D flip-flop is true MFM and is sent to a delay line. A delay line determines the amount of delay for precompensation. No delay corresponds to shifting the data bit early; the first tap is approximately 12 ns of delay and is the "normal", or no delay and the second tap provides 12 ns of delay, referenced to the "normal" write data. Which output is selected is determined by the states on RWC, $\overline{Early}$ and $\overline{Late}$. This function was generated with a 74s151 multiplexer. When RWC is inactive $\overline{EARLY}$ and $\overline{LATE}$ only select "normal" data since they are always active. The precompensated write data is then driven onto the data cable by an RS-422 driver.

Read Data Logic

The PLL generates the RD CLOCK that is used to decode the serial MFM data from the drive. A selected drive issues read data, unless WR GATE is active. A one-shot generates a pulse of 220–270 ns to provide the DRUN input. Only during an all zero's or one's field will the DRUN input stay high, as it will be retriggered every 200 ns (the minimum distance that separates continuous clock and data bits). As soon as DRUN is determined to be valid, the RD GATE output will go active, switching the PLL from the 10 MHz local clock input to disk data. The PLL will synchronize to the incoming serial data and generate a Read Clock of the proper timing and phase. The 82062 will then start to search for the address mark which is indicated by DRUN going low at the address mark.

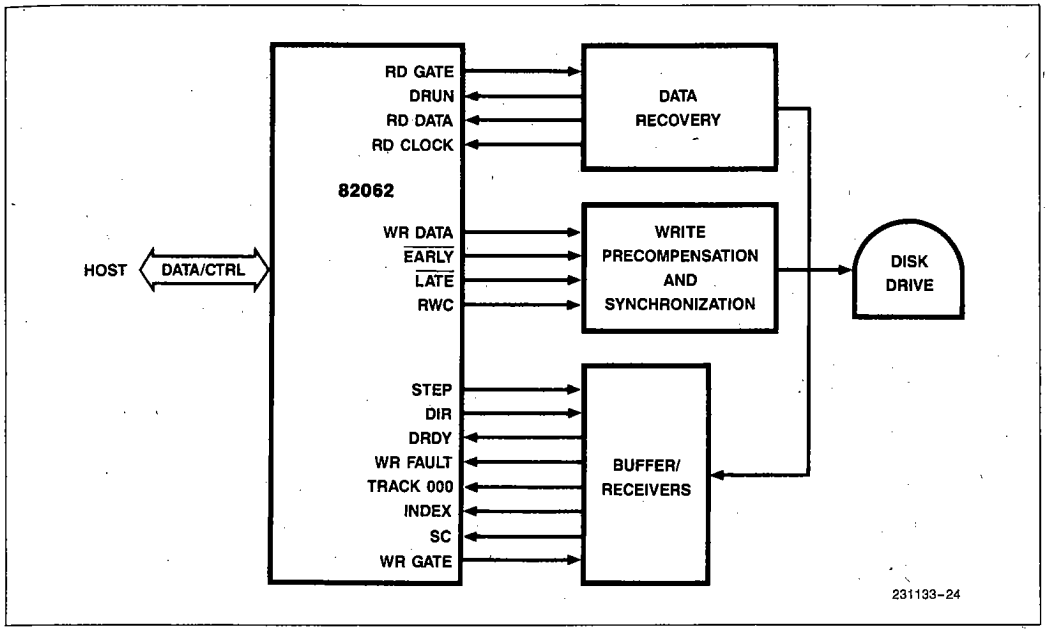


Figure 30. 82062 Disk Interface Block Diagram

No detail is provided herein on PLL design, as it is beyond the scope of this document. PLL design should be left to experienced designers, since minute changes in temperature and component values will drastically affect the soft error rate. As an alternative, several companies manufacture very high speed PLL chips for MFM encoded disk drives. Besides being fairly easy to design in, they reduce the number of components and board area needed for the sophisticated PLL.

Software Driver Overview

Presented in Appendix B is a listing of the software used to exercise the SBX 82062 board. Communication between the host software and the SBX driver routine is done through a structure located in system RAM. The host routine fills in required parameters, then passes the address of this communication block to the SBX driver routine. The driver routine pulls necessary values from this command block (CBL), executes a disk operation, then fills the CBL with the 82062's register contents, plus status and error information. The command block structure is shown in Figure 31.

Command	Byte
Rwc Reg	Byte
Sector Cnt.	Byte
Sector Num.	Byte
Cyl Low	Byte
Cyl High	Byte
SDH Reg	Byte
Status Reg	Byte
Error Reg	Byte
Host Buffer	Pointer

Figure 31

The host board did not have a DMA controller available, so an interrupt is issued from the BDRQ line and OR'ed with the 82062's interrupt line as interrupt sources were limited by the host. When an interrupt occurs, the interrupt procedure checks for either a data transfer, and executes it, or the completion of the command. If the interrupt signifies command completion, the interrupt procedure fills the command block with the 82062's task, status and error registers.

In this example, the host software examines one byte in the command block and until this byte is changed to a 00, no other command blocks will be passed to the disk driver routine. An alternative would be to issue a software interrupt to notify the microprocessor that the disk operation has finished and the command block contains parameters from the last operation and that a new disk command could start.

The driver for this example allows polling for non-data transfer commands, and must use interrupts for data transfers. As mentioned earlier, microprocessor intervention is required since the sector buffer is much larger than one sector and will not generate a BRDY. The microprocessor must write to an I/O port, which sets BRDY, after each host to sector buffer transfer. An actual software implementation would not include the polling and interrupt routines together, as only one method would generally be used.

The calling routine, which would normally be a directory program, places the values for which sector, number of sectors, etc., in the CBL. The disk routine is called and the address of this structure is passed on the stack. The disk driver places these parameters in the 82062's Task registers and initiates a command.

If the interrupt driven method was chosen, the disk driver routine returns to the calling routine. This permits other processing to be performed while the disk is executing a command. At some point, an interrupt will be generated, either from BRDY or INTRQ. Control will pass to the driver and the status register will be checked. If a data transfer is needed, either the microprocessor can transfer data or a DMA controller can perform the function. Once the transfer of data to the buffer is finished the microprocessor must set BRDY through an I/O port.

APPENDIX A
ST506 INTERFACE

THE ST506 INTERFACE

The ST506 interface is a modified version of Shugarts floppy disk drive interface and has been promoted by Seagate Technology. This interface is intended to be easy and low in cost to implement, yet provide a medium level of performance. The interface rigidly defines several areas: the hardware interconnects, the data transfer rate, the data encoding method, and how the disk is formatted.

Data Transfer Rate

The data transfer rate depends upon the linear bit density of the disk media and the speed at which the disk spins. ST506 specifies a 5 Mbit/second transfer rate. The typical ST506 drive has a nominal linear density of 10,416 bytes and a disk speed of 3600 rpm, which yields a 5 Mbit/second data transfer rate. No deviation from 5 M/bits second is allowed.

Increasing the linear density to increase storage capacity would require a decrease in disk speed. Otherwise, the data rate would increase. This decrease in disk speed would cause access times to increase, which many would deem unacceptable. To increase storage capacity, and remain ST506 compatible, either the number of cylinders and/or the number of platters can increase.

Data Encoding

ST506 requires that the serial data, sent between the drive and the controller, be encoded according to MFM rules. The basic unit of storage is a bit cell, which stores one bit infromation. This bit cell is divided into two halves, consisting of a clock bit and a data bit (see Figure A-1).

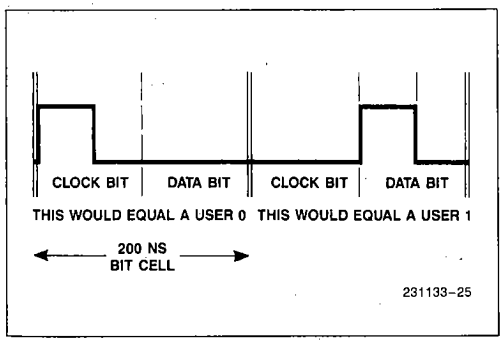


Figure A-1

The encoding rules for MFM are fairly simple:

1. A clock bit is written when the previous and the current bit cell does not contain a data bit.

2. A data bit is written whenever there is a "one" from the user.

Sync fields are composed of zeroes which generates a series of clock bits in the bit cell's. A phase lock loop locks on to the data stream during this period and generates a signal of the proper phase and frequency which is used to decode the combined clock and data serial data stream.

Disk Format

All disk media must be written with a specified format so that data may be reliably stored and retrieved. The smallest unit of controller accessible data is the sector which typically contains sync fields, ID fields, and a data field, and buffer fields.

The format of the disk required by ST506 is shown in Figure A-2. It should be noted that this format is fixed in the 82062. The user has options only for GAP1 and 3 length (when changing sector size or ECC) and whether to have 82062 CRC checking or user supplied ECC syndrome bits.

Gap 1 – Index Gap

Gap 1 serves two purposes. The first is to allow for variations in the index pulse timing due to motor speed variations. The second purpose is to allow a small delay to permit a different head to be selected without missing a sector. This is more of a data transfer optimization function and requires the disk controller to know which head is to be selected, when the last sector of a track has been read, and the next logical sector in the file exists on another platter. The 82062 does not switch heads automatically. Whether this scheme can be used or not depends upon the μP being able to alter one register in the 82062, before the next sector passes beneath the heads.

This gap is typically 12 bytes long and is written by the 82062 as 4E Hex.

Gap 2 – Write Splice Gap

This gap follows the CRC bytes of the ID field and continues up to the data field address mark. When up-

dating a previously written sector, motor speed variations could turn on the write coil, as the head was passing over the ID field. This gap prevents this from occurring. The value written is OOH and also serves as the PLL sync field for the data field. The minimum value is determined by the "lock up" performance of the PLL. The 82062 writes sixteen bytes for this field once WG is activated. The user has no control over this field.

Gap 3 – Post Data Field Gap

Gap 3 is very similar to Gap 2 as it is used as a speed tolerance buffer also. Without this gap, and with the motor speed varying slightly, it would be possible for the upcoming sector's sync field and ID field to be overwritten. This value is '4E' H and is typically 15 bytes long. The 82062's Gap 3 length is programmable. The exact value is dependent upon several factors. Refer to 82062 Format command, Software Section: General Programming Section.

Gap 4 – Track Buffer Gap

This gap follows the last sector on a track and is written until an index pulse is received. Its purpose is to prevent the last sector from overflowing past the index gap, and absorb track length variations when ECC is used (ECC uses more bytes than CRC). The value is '4E' H and is about 320 bytes when CRC and 256 byte sectors are used. The 82062 writes this field only during formatting. The user has no control over the number of bytes written with the 82062.

ID Fields

The controller uses ID fields to locate any individual sector. An address mark of two bytes precedes the ID field and the data field in a sector. An address mark tells the controller the nature of the upcoming information. ID fields are used by the disk controller and are not passed to the host.

Sector Interleaving

Sector interleaving occurs when logical sectors are in a non-sequential order, which is determined during formatting. An advantage is that there is a delay between logically sequential sectors. This delay can be used for data processing and then deciding if the next sector should be read. Without interleaving, the next sector could slip by, imposing a one revolution delay (approx. 16.7 ms). An additional benefit to this delay is that bus utilization is reduced by spreading the data transfer over a greater amount of time. The delay between sectors can be determined as follows:

$$\frac{\text{1 Revolution Period}}{\text{Sectors/Track}} \times (\text{Interleave factor} - 1) = \text{Delay}$$

For the typical ST506 drive with four-way interleaving this yields 1.57 ms of delay.

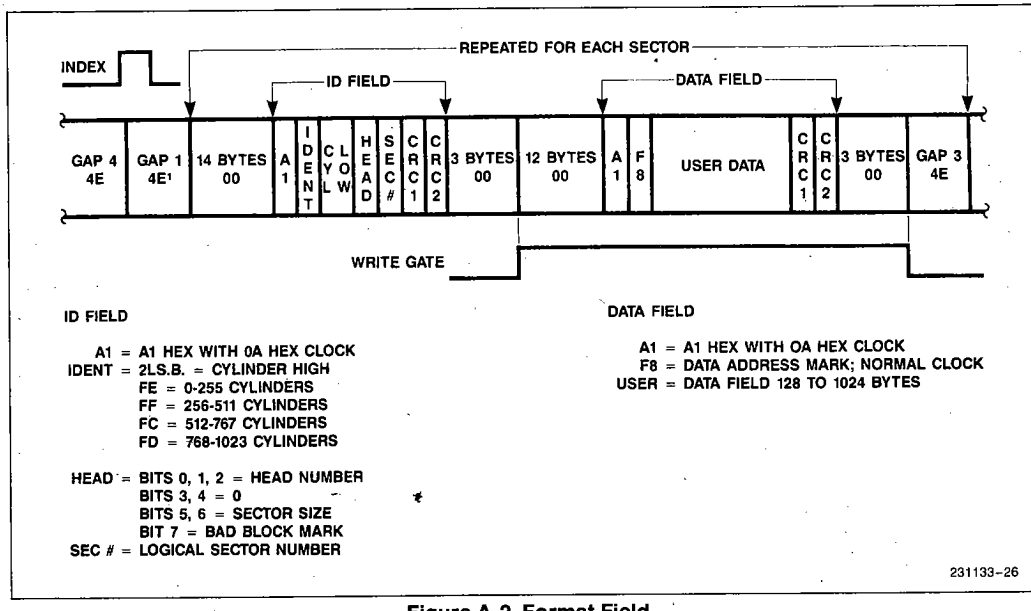


Figure A-2. Format Field

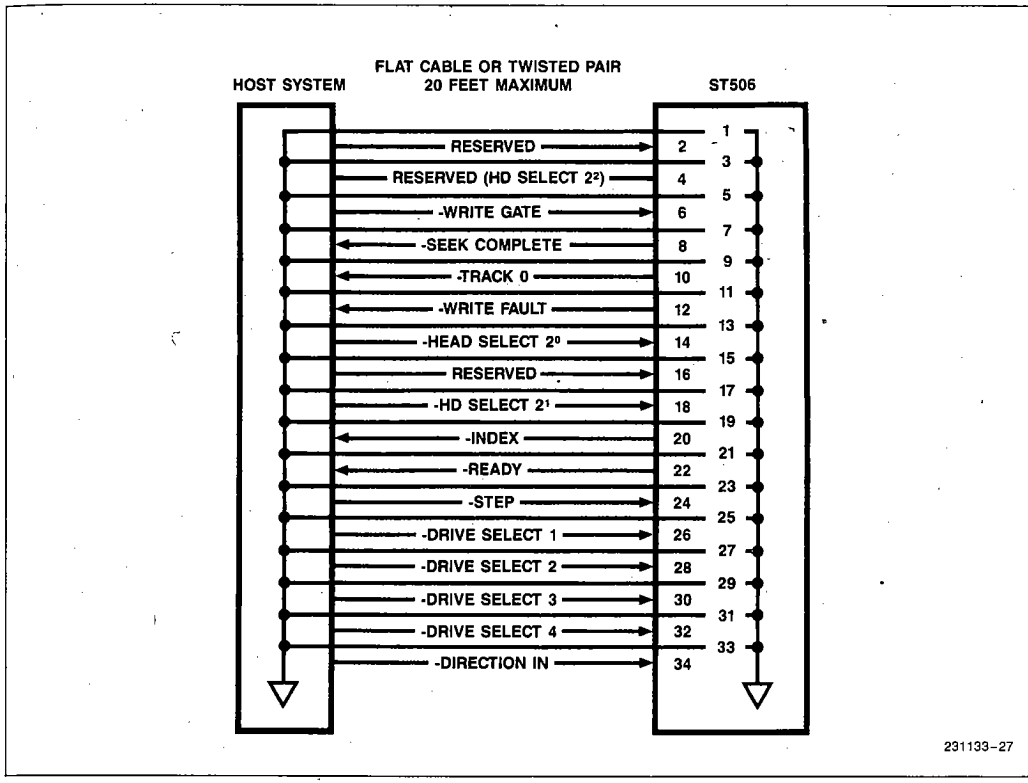


Figure A-3

The disadvantage to interleaving is that file transfers take longer, which may slow down the overall system. A four-way interleaved disk will have the transfer rate reduced to an average of 1.25 Mbit/sec.

The 82062 leaves the logical sector sequence to the user.

ELECTRICAL INTERFACE

The interface to the ST506 drive is divided into three categories and they are:

1. control signals,
2. data signals,
3. power.

Control Signals

The functions of the control signals are not covered in detail here. Their purpose can be found in the pin descriptions section. All control lines are digital in nature and either provide signals to the drive or inform the host of certain conditions. A diagram of the 34 pin control connector is shown in Figure A-3.

The driver/receiver logic diagram is shown in Figure A-4 and the electrical characteristics are:

	Voltage	Current
True	0.0 VDC to 0.4 VDC	−40 mA (IOL max.)
False	2,5 VDC to 5.25 VDC	250 μA (IOH open)

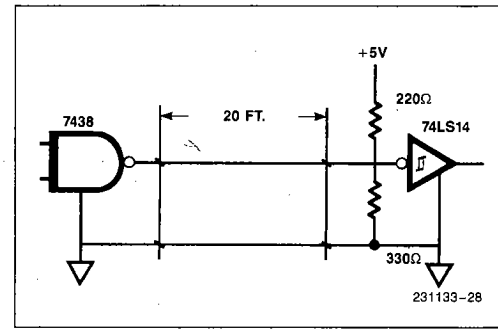


Figure A-4

Data Signals

The lines associated with the transfer of read/write data between the host and the drive are differential in nature and may not be multiplexed between drives. There is one pair of balanced lines for each read and write data line per drive and must conform to the RS-422 specification. Figure A-5 shows the receiver/transmitter combination.

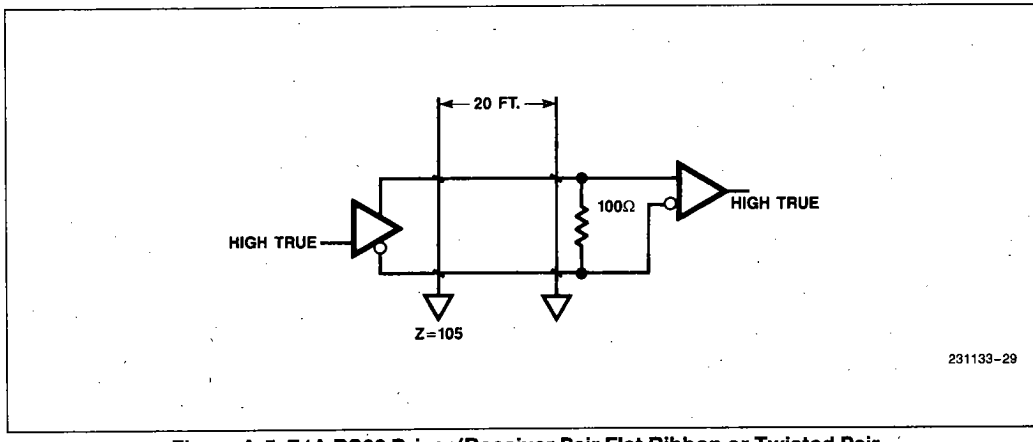


Figure A-5. E1A RS22 Driver/Receiver Pair Flat Ribbon or Twisted Pair

APPENDIX B
SOFTWARE DRIVER

```
SERIES-III PL/M-86 V2.3 COMPILATION OF MODULE DISK_IO_MODULE
OBJECT MODULE PLACED IN :F2:DISKIO.OBJ
COMPILER INVOKED BY:  PLM86.86 :F2:DISKIO.P86


                $TITLE('82062/SBX DISK CONTROLLER')
  1             DISK_IO_MODULE: DO;

                /*  CBL_PTR IS A POINTER TO A COMMAND BLOCK- HENCE CBL.
                    THIS COMMAND BLOCK RESIDES IN RAM AND CONTAINS ALL
                    VALUES REQUIRED BY THIS PROGRAM TO OPERATE THE 82062
                    DISK CONTROLLER.  ONCE THIS PROCEDURE IS CALLED, THE
                    CBL IS REMAIN UNTOUCHED UNTIL THE COMMAND BYTE IS
                    SET TO A 00 VALUE.  THIS ROUTINE WILL CALL THE CALLING
                    PROGRAM WHEN A COMMAND IS COMPLETED.

                    REV             DATE         NAME         DESCRIPTION

                    1.0          1/JUL/84   J. SLEEZER      INITIAL
                */


                /*        PROGRAM CONSTANTS              */

  2   1         DECLARE LIT             LITERALLY  'LITERALLY',
                        TRUE            LIT        '0FFH',
                        FALSE           LIT        '00H',
                        FOREVER         LIT        'WHILE TRUE';


                /*        BOARD ADDRESSING    FOR THE 86/05        */

  3   1         DECLARE BASE_ADDR       LIT        '80H',
                        SCTR_BFFR       LIT        'BASE_ADDR',
                        ERR_REG         LIT        'BASE_ADDR + 02H',   /* READ ONLY */
                        SEC_CNT_REG     LIT        'BASE_ADDR + 04H',
                        SEC_NUM_REG     LIT        'BASE_ADDR + 06H',
                        CYL_LOW_REG     LIT        'BASE_ADDR + 08H',
                        CYL_HI_REG      LIT        'BASE_ADDR + 0AH',
                        S_DR_HD_REG     LIT        'BASE_ADDR + 0CH',
                        STATUS_REG      LIT        'BASE_ADDR + 0EH',   /* READ ONLY */
                        COMMAND_REG     LIT        'BASE_ADDR + 0EH',   /* WRITE ONLY */
                        WR_PCMP_REG     LIT        'BASE_ADDR + 02H',   /* WRITE ONLY */
                        BFFR_RESET      LIT        '92H',
                        BFFR_RDY        LIT        '94H',
                        SEC_BUF         LIT        '2048';


                /*******        82062 COMMANDS      ********/

  4   1         DECLARE RESTORE         LIT        '1FH',
                        SEEK            LIT        '7FH',
                        FORMAT          LIT        '50H',
                        SCAN_ID         LIT        '40H',
                        READ_SEC        LIT        '20H',
                        WRITE_SEC       LIT        '30H',
                        ECC_EN          LIT        '80H',   /* TO BE OR'D WITH VALUE IN SDH REG */
                        NO_INTERPT      LIT        '00H',
                        INTR_ON_CMD     LIT        '08H',
                        MULT_SCTR       LIT        '04H';
```

231133-36

```
                /*      STATUS REGISTER BITS        */

  5   1         DECLARE ERR             LIT     '01H',
                        CIP             LIT     '02H',
                        DRQ             LIT     '80H',
                        SC              LIT     '10H',
                        WRF             LIT     '20H',
                        DRDY            LIT     '40H',
                        BUFBSY          LIT     '80H';  /* USER WILL NEVER SEE THIS BIT SET */


                /*      PROGRAM VARIABLES           */

  6   1         DECLARE CMD_BLOCK_PTR   POINTER,
                        CBL             BASED           CMD_BLOCK_PTR STRUCTURE (
                                        COMMAND         BYTE,
                                        PRECMP          BYTE,
                                        S_CNT           BYTE,
                                        SCTR            BYTE,
                                        LOW_CYL         BYTE,
                                        HI_CYL          BYTE,
                                        SDH             BYTE,
                                        STATUS          BYTE,
                                        ERRS            BYTE,
                                        INTERUPT        BYTE,
                                        RET_PROC        POINTER,
                                        BUFF_PTR        POINTER);

  7   1         DECLARE BUFFER_PTR      POINTER,
                        BUFF            BASED   BUFFER_PTR (1) BYTE,
                        STATUS          BYTE,
                        ERRORS          BYTE,
                        COMMAND         BYTE;



                $EJECT




                /*************************************/

                /*      82062 POLL ROUTINE          */

                /*************************************/

  8   1         POLL: PROCEDURE;

  9   2         DECLARE COUNT DWORD;

 10   2             COUNT = 7FFFFH;     /* LOOP FAILSAFE - TWEAK AS REQUIRED */
 11   2             STATUS = INPUT(STATUS_REG);
 12   2             DO WHILE ((STATUS AND (CIP OR DRDY)) = (CIP OR DRDY));
 13   3                 IF COUNT = 00 THEN RETURN;
 15   3                 IF (STATUS AND DRQ) = DRQ THEN DO;
 17   4                     CALL XFER_DATA;
 18   4                 END;
 19   3                 STATUS = INPUT(STATUS_REG);
 20   3                 COUNT = COUNT - 1;
 21   3             END;            /* IF THE ROUTINE EXPIRES DUE TO COUNT = 0, ALL DISK */
                                    /* REG VALUES IN THE CBL WILL CONTAIN THE STATUS REG */
                                    /* WHICH WILL = A BUSY PATTERN AND CBL.COMMAND WILL  */
                                    /* CONTAIN 00, INDICATING THE COMMAND IS FINISHED    */
 22   2         END POLL;

                /*********************************************************************/

                /*      TRANSFER DATA BETWEEN HOST RAM AND ONBOARD SECTOR BUFFER     */

                /*********************************************************************/

 23   1         XFER_DATA: PROCEDURE;

 24   2         DECLARE CNT             BYTE,
                        INDEX           WORD,
                        SZ1             BYTE,
                        SECTR_SZ        WORD;
```

231133-37

```
25   2          SZ1 = (SHR(CBL.SDH,5) AND 03H);      /* OBTAIN SECTOR SIZE BITS  FROM SDH */
26   2          IF SZ1 = 00 THEN SECTR_SZ = 256;     /* REGISTER */
28   2          ELSE IF SZ1 = 01 THEN SECTR_SZ = 512;
30   2          ELSE IF SZ1 = 02 THEN SECTR_SZ = 1024;
32   2          ELSE IF SZ1 = 03 THEN SECTR_SZ = 128;

34   2          IF CBL.SDH AND ECC_EN = ECC_EN THEN
35   2               SECTR_SZ = SECTR_SZ + 7;

36   2          IF(((CBL.COMMAND AND OFOH = READ_SEC) OR(CBL.COMMAND AND OFOH = WRITE_SEC))
37   2              AND (CBL.COMMAND AND OFH = MULT_SCTR)) THEN DO;/* VARIOUS SECTOR SIZES*/
38   3              CNT = (SEC_BUF/CBL.S_CNT);              /* ARE POSSIBLE.  THIS FIGURES  */
39   3              DO WHILE (CNT * SECTR_SZ) > SEC_BUF;    /* HOW MANY SECTORS WILL FIT    */
40   4                  CNT = CNT - 1;                      /* INTO THE BOARDS SECTOR BFFR */
41   4              END;
42   3              SECTR_SZ = SECTR_SZ * CNT;
43   3          END;

               /*  OUTPUT(BFFR_RESET) = 00;  */

44   2          IF (SHR(CBL.COMMAND,4) AND 03H) = 02H THEN DO;   /*  READ COMMAND */
46   3              DO INDEX = 0 TO (SECTR_SZ - 1);
47   4                  BUFF(INDEX) = INPUT(SCTR_BFFR);
48   4              END;
49   3          END;
50   2          ELSE DO;                              /*  WRITE  OR FORMAT COMMAND */
51   3              DO INDEX = 0 TO (SECTR_SZ - 1);
52   4                  OUTPUT(SCTR_BFFR) = BUFF(INDEX);
53   4              END;
54   3          END;

55   2          OUTPUT(BFFR_RDY) = 00;  /* ACTIVATES 062'S BRDY LINE */

56   2      END XFER_DATA;

               $EJECT




               /************************************/

               /*       UPDATE COMMAND BLOCK         */

               /************************************/

57   1      UPDATE_CBL:  PROCEDURE;

58   2          CBL.S_CNT = INPUT(SEC_CNT_REG);
59   2          CBL.SCTR = INPUT(SEC_NUM_REG);
60   2          CBL.LOW_CYL = INPUT(CYL_LOW_REG);
61   2          CBL.HI_CYL = INPUT(CYL_HI_REG);
62   2          CBL.SDH = INPUT(S_DR_HD_REG);
63   2          CBL.STATUS = STATUS;
64   2          CBL.ERRS = INPUT(ERR_REG);

65   2      END UPDATE_CBL;

               /************************************/

               /*       WRITE THE CBL TO 82062       */

               /************************************/

66   1      WR_CBL:  PROCEDURE;

67   2          OUTPUT(WR_PCMP_REG) = CBL.PRECMP;
68   2          OUTPUT(SEC_CNT_REG) = CBL.S_CNT;
69   2          OUTPUT(SEC_NUM_REG) = CBL.SCTR;
70   2          OUTPUT(CYL_LOW_REG) = CBL.LOW_CYL;
71   2          OUTPUT(CYL_HI_REG) = CBL.HI_CYL;
72   2          OUTPUT(S_DR_HD_REG) = CBL.SDH;

73   2      END WR_CBL;

               $EJECT
```

231133-38

```
                /*****************************************/
                /*****      MAIN PROGRAM          **********/

 74   1         DISK: PROCEDURE(CBL_PTR) PUBLIC;

 75   2         DECLARE CBL_PTR POINTER;

 76   2             CMD_BLOCK_PTR = CBL_PTR;                    /* ADDRESS OF STRUCTURE */
 77   2             BUFFER_PTR = CBL.BUFF_PTR;                  /* THAT CONTAINS 82062  */
                                                                 /* TASK REG DATA  */

 78   2             CALL WR_CBL;                                /* A DUMMY COMMAND TO READ*/
 79   2             IF CBL.COMMAND = 99H THEN DO;               /*THE CURRENT REG VALUES  */
 81   3                 CALL UPDATE_CBL;
 82   3                 CBL.COMMAND = 00;
 83   3                 CBL.STATUS = INPUT(STATUS_REG);
 84   3                 RETURN;
 85   3             END;

 86   2             IF (INPUT(STATUS_REG) AND DRDY) <> DRDY THEN DO; /* NO COMMAND IS ISSUED */
 88   3                 CBL.STATUS = INPUT(STATUS_REG);             /* IF THE 82062 SEES    */
 89   3                 CBL.COMMAND = 00H;                          /* THAT THE SELECTED    */
 90   3                 RETURN;                                     /* DRIVE IS NOT READY   */
 91   3             END;

 92   2             OUTPUT(BFFR_RESET) = 00H;

 93   2             IF (CBL.COMMAND AND 0F0H) = READ_SEC THEN    /* FOR PROGRAM CONSISTENCY */
 94   2                 CBL.COMMAND = CBL.COMMAND OR INTR_ON_CMD; /* SET INTERUPT FOR COMMAND*/
                                                                 /* TERMINATION */

 95   2             OUTPUT(COMMAND_REG) = CBL.COMMAND;       /* A DELAY IS NEEDED BECAUSE FAST*/
 96   2             CALL TIME(100);                          /* UP'S CAN READ THE STATUS REG  */
 97   2             IF CBL.INTERUPT = NO_INTERPT THEN DO;    /* BEFORE A VALID STATUS IS READY*/
 99   3                 CALL POLL;
100   3                 CALL UPDATE_CBL;
101   3                 CBL.COMMAND = 00;
102   3                 RETURN;
103   3             END;

104   2         END DISK;

                $EJECT


                /*****************************************/

                /*       INTERRUPT SERVICE ROUTINE       */

                /*****************************************/

105   1         DISK_SERVICE: PROCEDURE PUBLIC;

106   2             CALL TIME(500);
107   2             STATUS = INPUT(STATUS_REG);

108   2             IF (STATUS AND CIP) = 00  THEN DO;
110   3                 CALL UPDATE_CBL;
111   3                 CBL.COMMAND = 00;
112   3                 OUTPUT(BFFR_RESET) = 00H;
113   3                 RETURN;
114   3             END;
115   2             ELSE CALL XFER_DATA;

116   2         END DISK_SERVICE;

117   1         END DISK_IO_MODULE;

MODULE INFORMATION:

     CODE AREA SIZE     = 02EEH    750D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0011H     17D
     MAXIMUM STACK SIZE = 000AH     10D
     272 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS
```

231133-39

```
DICTIONARY SUMMARY:

     31KB MEMORY AVAILABLE
     5KB MEMORY USED    (16%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION



SERIES-III PL/M-86 V2.3 COMPILATION OF MODULE HOST_MODULE
OBJECT MODULE PLACED IN :F2:DSKHST.OBJ
COMPILER INVOKED BY:  PLM86.86 :F2:DSKHST.P86


                $TITLE('DEMO PROGRAM FOR SBX062')
  1             HOST_MODULE:DO;

                /*  PROGRAM TO EXERCISE THE 82062/SBX BOARD  USING THE 957 MONITOR
                 ON AN SBC 86/05.  THIS PROGRAM DEMONSTRATES HOW THE DISKIO MODULE
                 IS USED.  THE CASE STATEMENTS IN THE MAIN SECTION SHOW THE VARIOUS
                 ROUTINES.  THE TYPICAL ROUTINES LIKE HEX TO ASCII, ETC., WERE
                 NOT INCLUDED IN THIS LISTING.  SEVERAL OF THE ROUTINES USE
                 STATEMENTS THAT COULD BE REDUCED CONSIDERABLY BUT WERE LEFT
                 SIMPLIFIED SO THAT ALL WOULD UNDERSTAND.

                    REV             DATE          NAME          DESCRIPTION

                    1.0           20/JUL/84    J. SLEEZER      INITIAL
                */


                /*          EXTERNAL ROUTINES          */

  2   1         CO:  PROCEDURE(CHAR) EXTERNAL;
  3   2              DECLARE CHAR BYTE;
  4   2         END CO;


  5   1         CI:  PROCEDURE BYTE EXTERNAL;
  6   2         END CI;

  7   1         DISK:PROCEDURE(CMD_BLK_PTR) EXTERNAL;/* THIS ROUTINE STARTS A DISK OPERATION */
  8   2             DECLARE CMD_BLK_PTR POINTER;
  9   2         END DISK;

 10   1         DISK_SERVICE: PROCEDURE EXTERNAL;/* THIS ROUTINE SERVICES THE 82062 INTERUPTS*/
 11   2         END DISK_SERVICE;


                /*          PROGRAM CONSTANTS             */

 12   1         DECLARE LIT             LITERALLY  'LITERALLY',
                        TRUE            LIT        '0FFH',
                        FALSE           LIT        '00H',
                        FOREVER         LIT        'WHILE TRUE',
                        SPACE           LIT        '20H',
                        CR              LIT        '0DH',
                        LF              LIT        '0AH',
                        RUB             LIT        '7FH',
                        BACKSP          LIT        '08H',
                        ESC             LIT        '1BH';


                /*       82062 COMMANDS               */
```

231133-40

```
13   1      DECLARE RESTORE          LIT     '10H',
                    SEEK             LIT     '70H',
                    FORMAT           LIT     '50H',
                    SCAN_ID          LIT     '40H',
                    READ_SECT        LIT     '28H',   /* INTR ONLY ON COMMAND TERMINATION */
                    WRITE_SCT        LIT     '30H',
                    MULT_SCTR        LIT     '04H',   /* TO BE OR'D WITH COMMAND */
                    NO_RETRYS        LIT     '01H',   /* TO BE OR'D WITH COMMAND */
                    NO_CRC           LIT     '80H',   /* TO BE OR'D WITH VALUE IN SDH REG */

                    P_COMP           LIT     '0',   /*  INDEXING INTO DISK_REG ARRAY */
                    SEC_CNT          LIT     '1',
                    SECTOR           LIT     '2',
                    CYL_LB           LIT     '3',
                    CYL_HB           LIT     '4',
                    SDH              LIT     '5';

             /*     STATUS REGISTER BITS          */

14   1      DECLARE ERR              LIT     '01H',
                    CIP              LIT     '02H',
                    DRQ              LIT     '80H',
                    SC               LIT     '10H',
                    WRF              LIT     '20H',
                    DRDY             LIT     '40H',
                    BUFBSY           LIT     '80H';   /* USER WILL NEVER SEE THIS BIT SET */

             /*     ERROR REGISTER BITS           */

15   1      DECLARE VALID_BITS       LIT     '0D7H',
                    AM_NT_FND        LIT     '001H',
                    TK00_ERR         LIT     '002H',
                    ABRTD_CMD        LIT     '004H',
                    ID_NT_FND        LIT     '010H',
                    DATA_ERR         LIT     '040H',
                    BAD_BLK          LIT     '080H';

             /*****      PROGRAM VARIABLES        *****/

16   1      DECLARE CMD_BLK(1)       STRUCTURE ( COMMAND          BYTE,
                                                 PRECMP           BYTE,
                                                 S_CNT            BYTE,
                                                 SCTR             BYTE,
                                                 LOWB_CYL         BYTE,
                                                 HIB_CYL          BYTE,
                                                 SDHD             BYTE,
                                                 STATUS           BYTE,
                                                 ERRS             BYTE,
                                                 INTERUPT         BYTE,
                                                 RET_PROC         POINTER,
                                                 BUFF_PTR         POINTER);

17   1      DECLARE COUNT                 WORD,

                    CHAR                  BYTE,
                    ERRORS                BYTE,
                    COMMAND               BYTE,
                    STEP_RATE             BYTE,
                    I                     WORD,
                    I2                    BYTE,
                    BUFFER(1100)          BYTE,
                    INDEX                 WORD,
                    DISK_IS_NOT_BUSY      BYTE,
                    TRACKS                BYTE,
                    PLATTERS              BYTE,
                    PLAT_CNT              BYTE,
                    TRACK_CNT             BYTE,
                    I_FACTOR              BYTE,
                    FRMT_BFFR_SIZE        BYTE,
                    LOG_SECT_NUM          BYTE,
                    MAKING_TABLE          BYTE,
                    AA                    BYTE,
                    INDX                  BYTE;

18   1      DECLARE DISK_REGS (6)         BYTE;

             $NOLIST
             $EJECT
```

231133-41

```
                /*****************************************/

                /*********      MAIN PROGRAM         *******/

                /*****************************************/

337   1         MAIN: DO;

338   2             STEP_RATE = 0FH;
339   2             PLAT_CNT =0FFH;
340   2             TRACK_CNT =0FFH;
341   2             PLATTERS = 00;
342   2             TRACKS = 00;
343   2             DO I = 0 TO 5;
344   3                 DISK_REGS(I) = 00;
345   3             END;
346   2             DISK_REGS(P_COMP) = 0FFH;
347   2             CALL UPDATE_CMD_BLK;
348   2             CMD_BLK(INDX).INTERUPT = 00;
349   2             CALL WRITEA(@(LF,LF,LF,LF,LF,00));
350   2             CALL WRITEA(@SIGN_ON);
351   2             OUTPUT(0C2H) = 0FDH;   /*  PERMITS AN INTERRUPT 1  */
352   2             INDX = 0;
353   2             CMD_BLK(INDX).BUFF_PTR = @BUFFER;
354   2             CALL SET$INTERRUPT(21H,CHECK_DISK);
355   2             ENABLE;
356   2             DISK_IS_NOT_BUSY = TRUE;

357   2             DO FOREVER;
358   3                 IF DISK_IS_NOT_BUSY THEN DO;
360   4                     CMD_BLK(INDX).COMMAND = 0FFH;
361   4                     CALL WRITEA(@(CR,LF,'COMMAND  >',00));
362   4                     CHAR = CI;
363   4                     CALL CO(CHAR);
364   4                     CALL CO(CR);
365   4                     CALL CO(LF);
366   4                     COMMAND = FALSE;
367   4                     I2 = 0;
368   4                     DO WHILE (COMMAND = FALSE);
369   5                         IF I2 > LENGTH(VALID_CMDS) THEN DO;
371   6                             CALL WRITEA(@('INVALID COMMAND',CR,LF,00));
372   6                             I2 = 0;
373   6                             CHAR = CI;
374   6                         END;


375   5                         IF CHAR = VALID_CMDS(I2) THEN
376   5                             COMMAND = TRUE;
377   5                         I2 = I2 + 1;
378   5                     END;
379   4                     DO CASE (I2 - 1);

                                /*   CASE 0 - READ SECTOR */
380   5                         DO;
381   6                             CALL WRITEA(@('READ SECTOR COMMAND',CR,LF,LF,00));
382   6                             CALL WRITE_REGS;
383   6                             DISK_IS_NOT_BUSY = FALSE;
384   6                             CMD_BLK(INDX).COMMAND = READ_SECT;
385   6                             CALL WRITEA(@('MULTIPLE SECTOR`S? >',00));
386   6                             CHAR = CI;
387   6                             IF CHAR = 'Y' THEN DO;
389   7                                 CALL WRITEA(@('YES - ',00));
390   7                                 CMD_BLK(INDX).COMMAND =
                                              CMD_BLK(INDX).COMMAND OR MULT_SCTR;
391   7                                 CALL WRITEA(@('DO NOT EXCEED BUFFER LIMIT !',CR,LF,00));
392   7                             END;
393   6                             ELSE CALL WRITEA(@('NO',CR,LF,00));

394   6                             CALL WRITEA(@('AUTOMATIC RETRIES? >',00));
395   6                             CHAR = CI;
396   6                             IF CHAR = 'N' THEN DO;
398   7                                 CALL WRITEA(@('NO',CR,LF,00));
399   7                                 CMD_BLK(INDX).COMMAND =
                                              CMD_BLK(INDX).COMMAND OR NO_RETRYS;
400   7                             END;
401   6                             ELSE CALL WRITEA(@('YES',CR,LF,00));

402   6                             CALL DISK(@CMD_BLK(INDX));
403   6                         END;                                    231133-42
```

```
                        /*   CASE 1 - WRITE SECTOR */
404   5                 DO;
405   6                     CALL WRITEA(@('WRITE SECTOR COMMAND',CR,LF,LF,00));
406   6                     CALL WRITE_REGS;
407   6                     CALL DATA_PAT;
408   6                     DISK_IS_NOT_BUSY = FALSE;
409   6                     CMD_BLK(INDX).COMMAND = WRITE_SCT;
410   6                     CALL WRITEA(@('MULTIPLE SECTOR'S? >',00));
411   6                     CHAR = CI;
412   6                     IF CHAR = 'Y' THEN DO;
414   7                         CALL WRITEA(@('YES - ',00));
415   7                         CMD_BLK(INDX).COMMAND =
                                    CMD_BLK(INDX).COMMAND OR MULT_SCTR;
416   7                         CALL WRITEA(@('DO NOT EXCEED BUFFER LIMIT !!',CR,LF,00));
417   7                     END;
418   6                     ELSE CALL WRITEA(@('NO',CR,LF,00));

419   6                     CALL WRITEA(@('ENABLE  RETRIES ? >',00));
420   6                     CHAR = CI;
421   6                     IF CHAR = 'N' THEN DO;
423   7                         CALL WRITEA(@('NO',CR,LF,00));
424   7                         CMD_BLK(INDX).COMMAND =
                                    CMD_BLK(INDX).COMMAND OR NO_RETRYS;
425   7                     END;
426   6                     ELSE CALL WRITEA(@('YES',CR,LF,00));

427   6                     CALL DISK(@CMD_BLK(INDX));
428   6                 END;

                        /*   CASE 2 - FORMAT TRACK */
429   5                 DO;
430   6                     CALL WRITEA(@('FORMAT TRACK',CR,LF,LF,00));
431   6                     CALL WRITE_REGS;
432   6                     DISK_IS_NOT_BUSY = FALSE;
433   6                     CMD_BLK(INDX).COMMAND = FORMAT;
434   6                     CALL WRITEA(@(' INTERLEAVE FACTOR? (1 TO ?)>',00));
435   6                     I_FACTOR = CI - '0';
436   6                     CALL CO(I_FACTOR + '0');
437   6                     CALL CO(CR);
438   6                     CALL CO(LF);
439   6                     FRMT_BFFR_SIZE = (2 * (CMD_BLK(INDX).S_CNT) + 1);
440   6                     DO I = 0 TO FRMT_BFFR_SIZE;
441   7                         BUFFER(I) = 00;
442   7                     END;
443   6                     LOG_SECT_NUM = 0;
444   6                     I= 1;
445   6                     MAKING_TABLE = TRUE;
446   6                     DO WHILE MAKING_TABLE;
447   7                         DO WHILE I <= FRMT_BFFR_SIZE;
448   8                             BUFFER(I) = LOG_SECT_NUM;
449   8                             LOG_SECT_NUM = LOG_SECT_NUM + 1;
450   8                             I = I +(I_FACTOR * 2);
451   8                         END;
452   7                         IF LOG_SECT_NUM < CMD_BLK(INDX).S_CNT THEN DO;
454   8                             I = I - (FRMT_BFFR_SIZE + 1);
455   8                             IF (I = 1) OR (BUFFER(I) <> 00) THEN
456   8                                 I = I + 2;
457   8                         END;
458   7                         ELSE MAKING_TABLE = FALSE;
459   7                     END;
460   6                     CALL WRITEA(@('256 TRACKS IS THE LIMIT',CR,LF,00));
461   6                     CALL WRITEA(@('HOW MANY TRACKS? IN HEX >',00));
462   6                     TRACKS = HEXIN(TRACKS);
463   6                     CALL CO(CR);
464   6                     CALL CO(LF);
465   6                     CALL WRITEA(@('HOW MANY SURFACES? I.E.,01 >',00));
466   6                     PLATTERS = HEXIN(PLATTERS);
467   6                     CALL CO(CR);
468   6                     CALL CO(LF);
469   6                     TRACK_CNT = 1;
```

231133-43

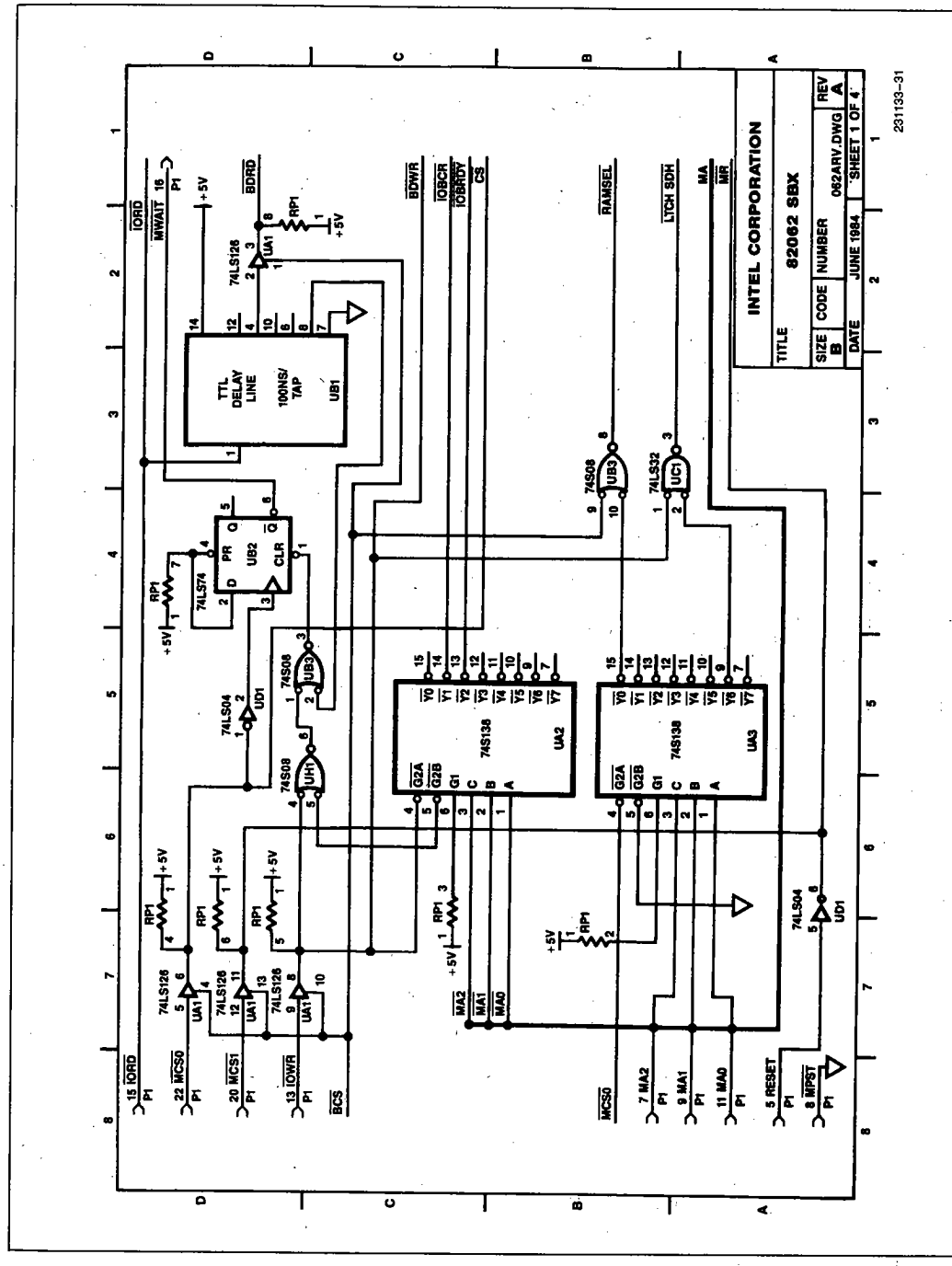
```
470   6                 DO WHILE TRACK_CNT <= TRACKS;
471   7                     PLAT_CNT = 1;
472   7                     DO WHILE PLAT_CNT <= PLATTERS;
473   8                         CALL UPDATE_CMD_BLK;
474   8                         CALL CO(CR);
475   8                         CALL WRITEA(@('TRACK = ',00));
476   8                         CALL DISP_HEX(@TRACK_CNT,1);
477   8                         CALL WRITEA(@('; HEAD = ',00));
478   8                         AA = DISK_REGS(SDH) AND 07H;
479   8                         CALL DISP_HEX(@AA,1);
480   8                         CMD_BLK(INDX).COMMAND = FORMAT;
481   8                         CALL DISK(@CMD_BLK(INDX));
482   8                         DO WHILE CMD_BLK(INDX).COMMAND <> 00;
483   9                         END;
484   8                         PLAT_CNT = PLAT_CNT + 1;
485   8                         DISK_REGS(SDH) = DISK_REGS(SDH) + 1;
486   8                     END;
487   7                     DISK_REGS(SDH) = DISK_REGS(SDH) - (PLATTERS);
488   7                     DISK_REGS(CYL_LB) = DISK_REGS(CYL_LB) + 1;
489   7                     IF DISK_REGS(CYL_LB) = 00 THEN
490   7                         DISK_REGS(CYL_HB) = DISK_REGS(CYL_HB) + 1;
491   7                     TRACK_CNT = TRACK_CNT + 1;
492   7                     CALL UPDATE_CMD_BLK;
493   7                 END;
494   6                 CALL CO(CR);
495   6                 CALL CO(LF);
496   6             END;

                      /*   CASE 3 - SCAN ID  */
497   5             DO;
498   6                 CALL WRITEA(@(' SCAN ID',CR,LF,LF,00));
499   6                 CALL WRITE_REGS;
500   6                 DISK_IS_NOT_BUSY = FALSE;
501   6                 CMD_BLK(INDX).COMMAND = SCAN_ID;
502   6                 CALL DISK(@CMD_BLK(INDX));
503   6             END;

                      /*   CASE 4 - SEEK TRACK */
504   5             DO;
505   6                 CALL WRITEA(@('SEEK TRACK',CR,LF,LF,00));
506   6                 CALL WRITE_REGS;
507   6                 CMD_BLK(INDX).COMMAND = SEEK OR STEP_RATE;
508   6                 DISK_IS_NOT_BUSY = FALSE;
509   6                 CALL DISK(@CMD_BLK(INDX));
510   6             END;
                      /*   CASE 5 - RESTORE */
511   5             DO;
512   6                 CALL WRITEA(@(' RESTORE COMMAND',CR,LF,LF,00));
513   6                 CALL WRITE_REGS;
514   6                 CMD_BLK(INDX).COMMAND = RESTORE OR STEP_RATE;
515   6                 DISK_IS_NOT_BUSY = FALSE;
516   6                 CALL DISK(@CMD_BLK(INDX));
517   6             END;

                      /*   CASE 6 - READ DISK REGISTER FILE */
518   5             DO;
519   6                 CALL WRITEA(@(' READ DISK REGISTERS',CR,LF,LF,00));
520   6                 CMD_BLK(INDX).COMMAND = 99H;
521   6                 CALL DISK(@CMD_BLK(INDX));
522   6                 CALL DISP_CMD_BLK;
523   6                 CMD_BLK(INDX).COMMAND = OFFH;
524   6             END;

                      /*   CASE 7 - HELP TABLE */
525   5             DO;
526   6                 CALL WRITEA(@HELP);
527   6                 CALL CO(LF);
528   6             END;

                      /*   CASE 8 - EXAMINE COMMAND BLOCK */
529   5             DO;
530   6                 CALL DISP_CMD_BLK;
531   6             END;
```

231133-44

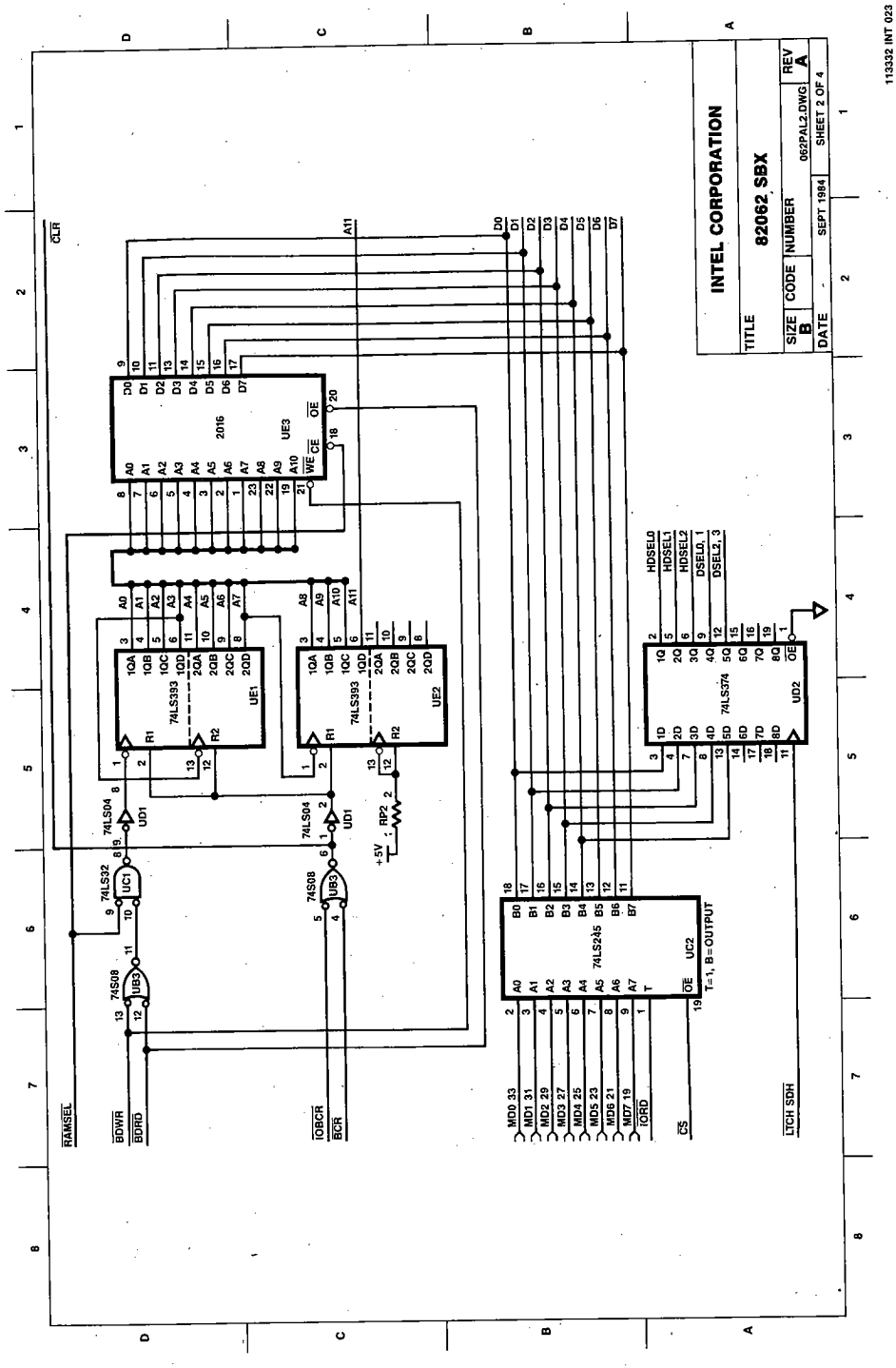
```
                                /*   CASE 9 - DISPLAY BUFFER DATA */
  532   5                       DO;
  533   6                           CALL WRITEA(@('DISPLAY ASCII<A> OR HEX<H> ? >',00));
  534   6                           CHAR = CI;
  535   6                           CALL CO(CHAR);
  536   6                           CALL CO(CR);
  537   6                           CALL CO(LF);
  538   6                           IF CHAR = 'A' THEN DO;
  540   7                               INDEX = 0;
  541   7                               DO WHILE CHAR <> ESC;
  542   8                                   DO I = 0 TO 255;
  543   9                                       CALL CO(BUFFER(INDEX + I));
  544   9                                   END;
  545   8                                   INDEX = INDEX + I;
  546   8                                   CHAR = CI;
  547   8                               END;
  548   7                           END;
  549   6                           IF CHAR = 'H' THEN DO;
  551   7                               INDEX = 0;
  552   7                               DO WHILE CHAR <> ESC;
  553   8                                   CALL DISP_HEX(@BUFFER(INDEX),256);
  554   8                                   INDEX = INDEX + 256;
  555   8                                   CHAR = CI;
  556   8                               END;
  557   7                           END;
  558   6                       END;

  559   5                   END; /* DO CASE */
  560   4               END;     /* IF */
  561   3               ELSE DO;
  562   4                   CALL WRITEA(@('*** DISK IS BUSY ***',CR,00));
  563   4               END;

  564   3               IF CMD_BLK(INDX).COMMAND = 00 THEN DO;
  566   4                   DISK_IS_NOT_BUSY = TRUE;
  567   4                   CALL DISP_STATUS;
  568   4               END;

  569   3           END;          /* FOREVER */

  570   2       END MAIN;
  571   1       END HOST_MODULE;
MODULE INFORMATION:

     CODE AREA SIZE     = 0FD8H   4056D
     CONSTANT AREA SIZE = 093CH   2364D
     VARIABLE AREA SIZE = 0480H   1152D
     MAXIMUM STACK SIZE = 003EH     62D
     868 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     31KB MEMORY AVAILABLE
     12KB MEMORY USED   (38%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

231133-45

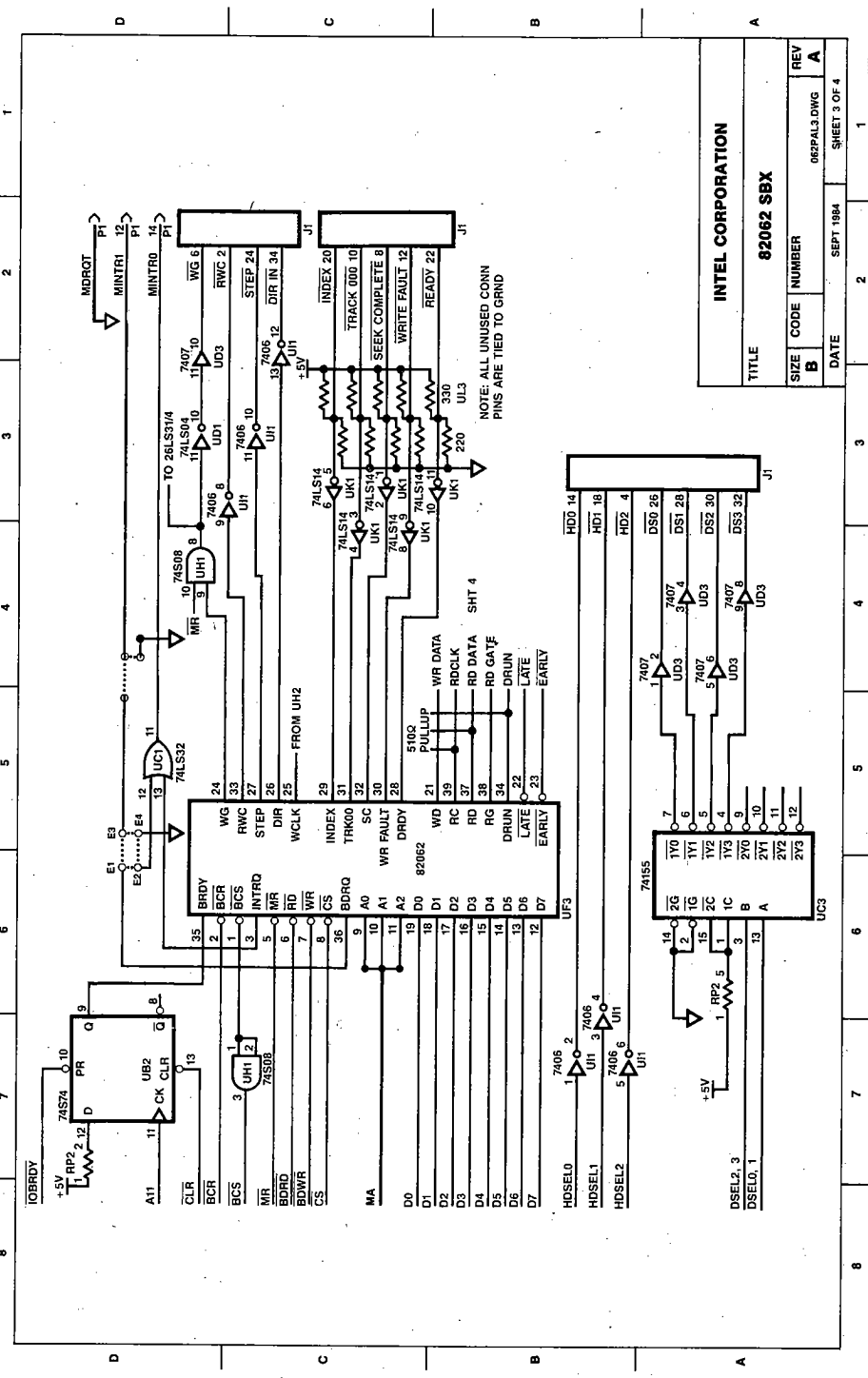
APPENDIX C
SCHEMATICS



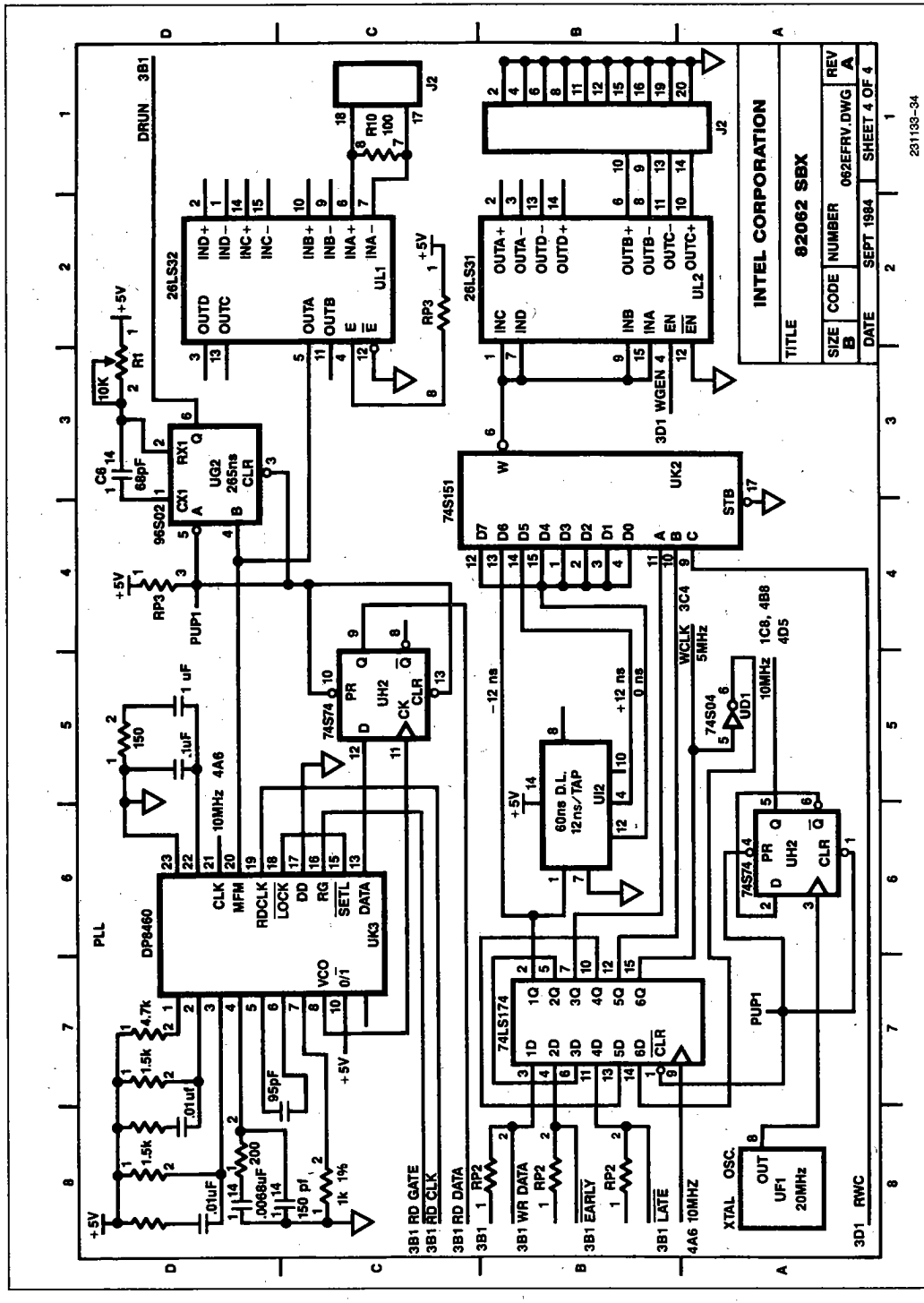
231133-002


INTEL CORPORATION
TITLE
82062 SBX
SIZE B
CODE
NUMBER
062PAL2.DWG
REV A
DATE
SEPT 1984
SHEET 2 OF 4
113332 INT 023
2016
UE3
74LS393
UE1
UE2
74LS04
UD1
74LS32
UC1
74S08
UB3
74LS374
UD2
74LS245
UC2
T=1, B=OUTPUT
RP2
+5V
CLR
RAMSEL
BDWR
BDRD
IOBCR
BCR
CS
LTCH SDH
HDSEL0
HDSEL1
HDSEL2
DSEL0_1
DSEL2_3
MD0 33
MD1 31
MD2 29
MD3 27
MD4 25
MD5 23
MD6 21
MD7 19
IORD

231133-002


INTEL CORPORATION
TITLE
82062 SBX
SIZE B
CODE
NUMBER
062PAL3.DWG
REV A
DATE
SEPT 1984
SHEET 3 OF 4
113333 INT 023
NOTE: ALL UNUSED CONN PINS ARE TIED TO GRND
TO 26LS31/4
FROM UH2
SHT 4
82062
74155
74S74
74LS32
74S08
74LS14
74LS04
7407
7406
510Ω PULLUP
IOBRDY
MDROT
MINTR1
MINTR0
WG
RWC
STEP
DIR IN
INDEX
TRACK 000
SEEK COMPLETE
WRITE FAULT
READY
WR DATA
RDCLK
RD DATA
RD GATE
DRUN
LATE
EARLY
HDSEL0
HDSEL1
HDSEL2
DSEL2, 3
DSEL0, 1

231133-002

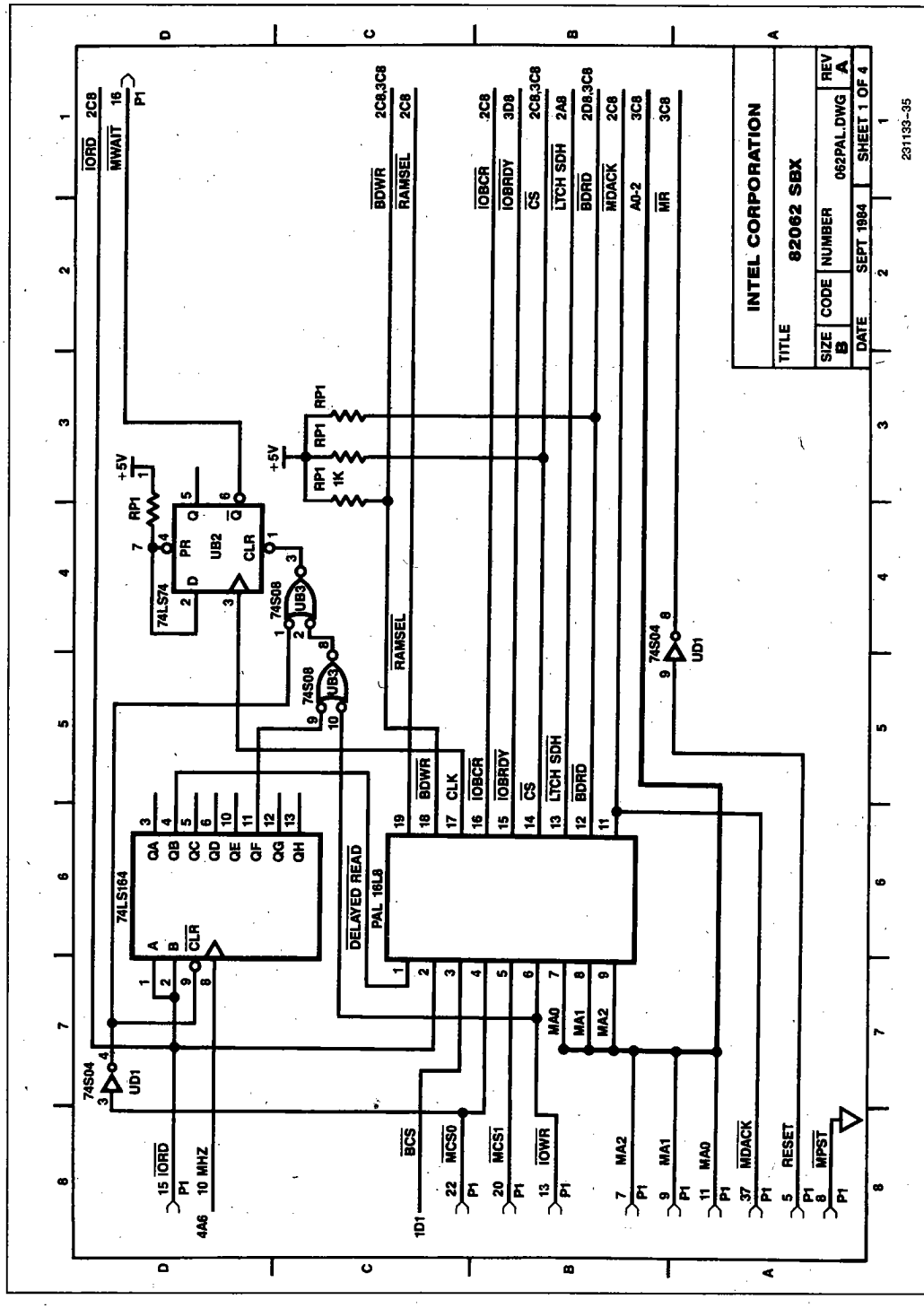

INTEL CORPORATION
TITLE 82062 SBX
SIZE B
CODE NUMBER 062EFRV.DWG
REV A
DATE SEPT 1984
SHEET 4 OF 4
231133-34
PLL
DP8460
UK3
74LS174
74S151
UK2
26LS32
UL1
26LS31
UL2
74S74
UH2
74S04
UG2
96S02
J2
DRUN 3B1
3D1 WGEN 4
WCLK 5MHz
1C8, 4B8
4D5
10MHz
60ns D.L.
12ns/TAP
UI2
+12 ns
0 ns
-12 ns
3C4
4A6
10MHz
XTAL OSC.
OUT
UF1
20MHz
4A6 10MHZ
3D1 RWC
3B1 RD GATE
3B1 RD CLK
3B1 RD DATA
3B1 WR DATA
3B1 EARLY
3B1 LATE
PUP1
+5V
10K
R1
C6
68pF
RP3
RP2
R10
100
4.7k
1.5k
.01uf
.01uF
.0068uF
200
150 pf
1k 1%
95pF
150
.1uF
1 uF

APPENDIX D

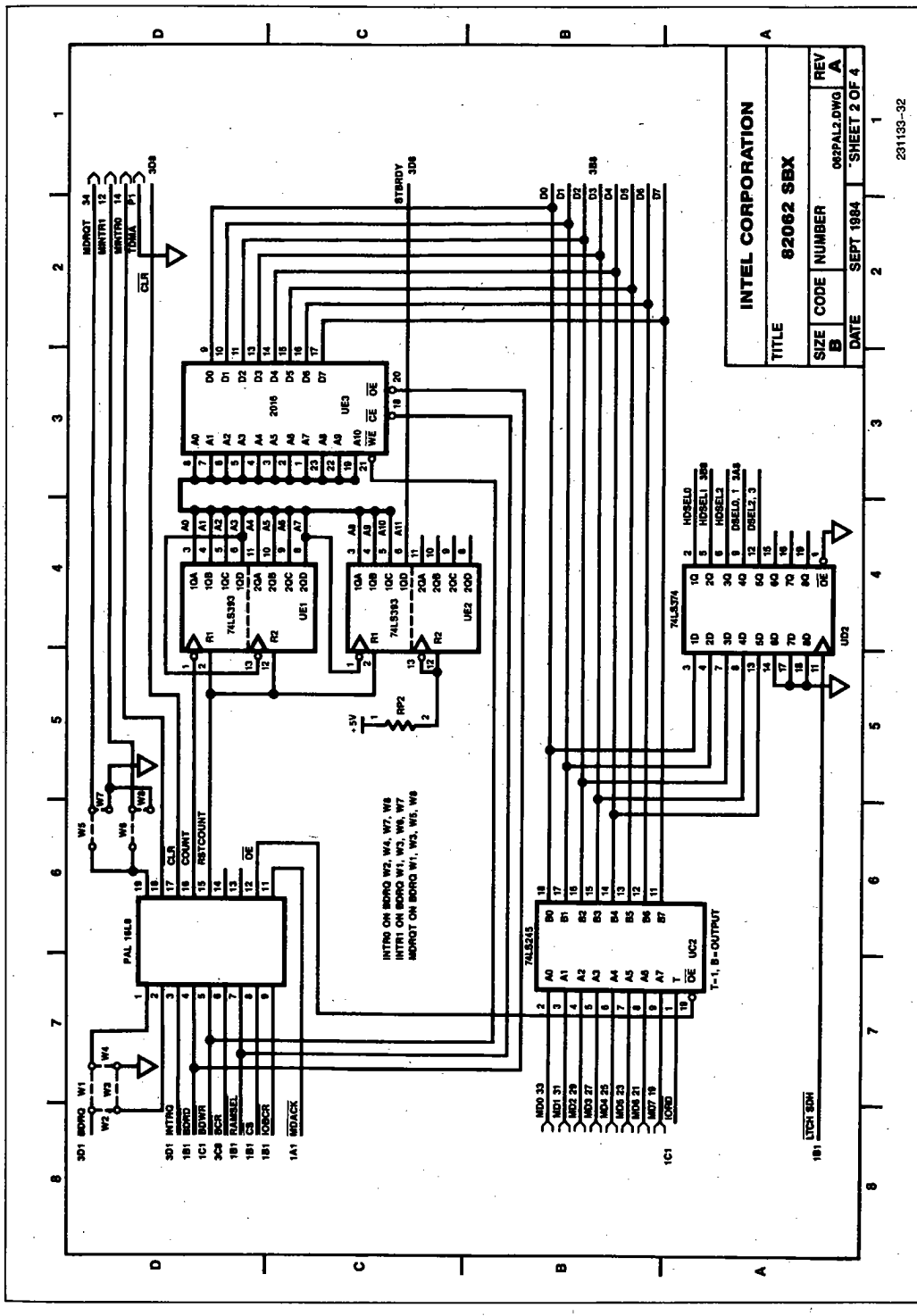
This appendix contains a schematic of the previous design using PAL's to replace the random logic. The previous design could not do DMA transfers and inserted a large delay when transferring data from buffer RAM to the system. The PAL version does do DMA transfers and buffer reads happen at full SBX bus speed. One other minor change was to replace the 500 ns delay line with a 74LS164, which is a more cost effective solution.

This schematic is only a paper design since only random logic was replaced with the PAL's.

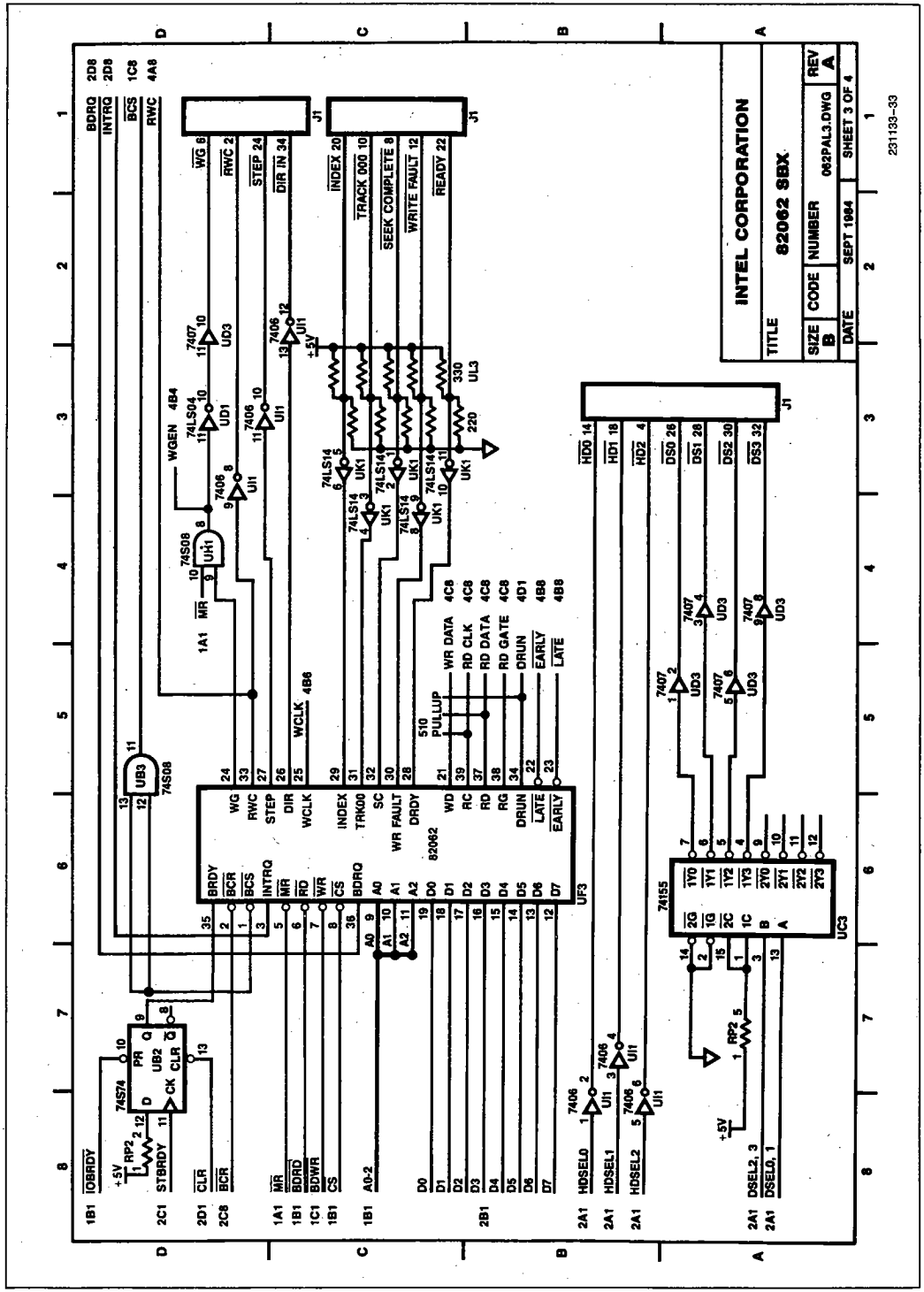
```
   PAL Equation's
PAL - Page 1:
   BDRD/ = (IORD/ * MDACK/) + (IORD/ * MCS0/ * MA0 * MA1 * MA2) +
(DELAYED-READ/ * CLK) IF BCS

   LTCHSDH/ = (MCS0/ * MA0/ * MA1 * MA2 * IOWR/)

   RAMSEL/ = (MCS0 * MA0 * MA1 * MA2) + (BCS/) + (MDACK/)

   IOBRDY/ = (MCS1/ * MA0/ * MA1 * MA2/ * IOWR/)

   IOBCR/ = (MCS1/ * MA0 * MA1/ * MA2/ * IOWR/)

   BDWR/ = (IOWR/) IF BCS

   CS/ = (MCS0/) IF BCS

   CLK = (MCS0/ * MA0 * MA1/ * MA2/) + (MCS0/ * MA0/ * MA1 * MA2/) + (MCS0/
* MA0 * MA1 * MA2/) + (MCS0/ * MA0/ * MA1/ * MA2) + (MCS0/ * MA0 * MA1/ *
MA2) + (MCS0/ * MA0/ * MA1 * MA2) + (MCS0/ * MA0 * MA1 * MA2)

PAL - Page 2:
   MINTR1/MDRQT = (PIN1)

   MINTR0 = (PIN2) + (INTRQ)

   COUNT = (BDWR/ + BDRD/) * (RAMSEL/)

   RSTCOUNT = (IOBCR/) + (BCR/)

   OE/ = (MDACK/) + (CS/)

   CLR/ = (IOBCR/) + (BCR/)
```


INTEL CORPORATION
TITLE 82062 SBX
SIZE B
CODE
NUMBER 062PAL.DWG
REV A
DATE SEPT 1984
SHEET 1 OF 4
231133-35
IORD 2C8
MWAIT 16
P1
BDWR 2C8,3C8
RAMSEL 2C8
IOBCR 2C8
IOBRDY 3D8
CS 2C8,3C8
LTCH SDH 2A8
BDRD 2D8,3C8
MDACK 2C8
A0-2 3C8
MR 3C8
+5V
RP1
1K
74LS74
UB2
74S08
UB3
74S04
UD1
74LS164
QA QB QC QD QE QF QG QH
A B CLR
DELAYED READ
PAL 16L8
CLK
MA0
MA1
MA2
BCS
MCS0
MCS1
IOWR
RESET
MPST
10 MHZ
4A6
1D1

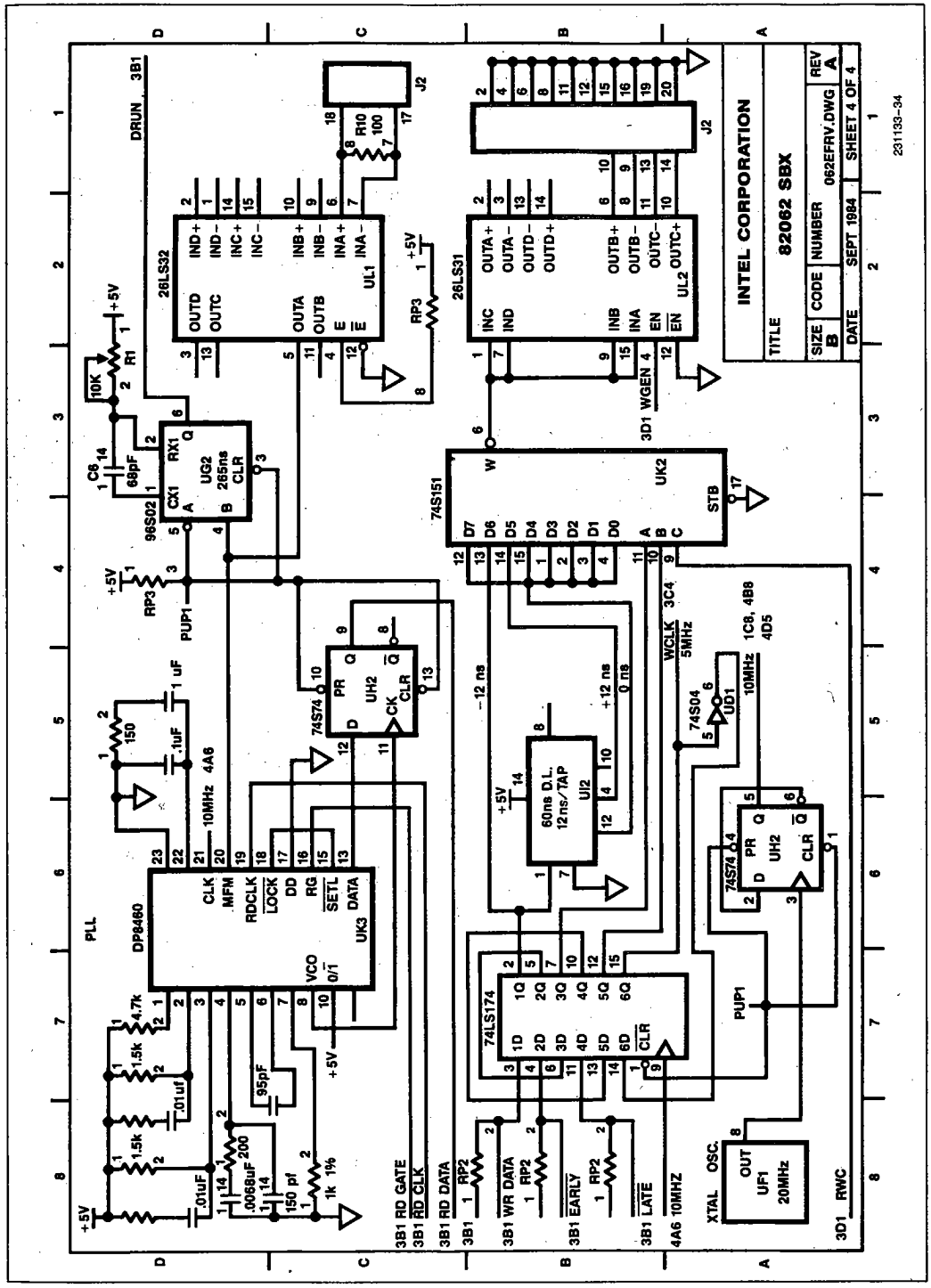
231133-002


INTEL CORPORATION
TITLE
82062 SBX
SIZE B
CODE
NUMBER
062PAL2.DWG
REV A
DATE
SEPT 1984
SHEET 2 OF 4
231133-32
INTRO ON BDRQ W2, W4, W7, W8
INTR1 ON BDRQ W1, W3, W6, W7
MDRQT ON BDRQ W1, W3, W5, W8
PAL 16L8
74LS393
74LS374
74LS245
T-1, B-OUTPUT
2016
+5V
RP2
MDRQT
MINTR1
MINTR0
TOMA
CLR
STARDY
BDRQ
INTRO
BDWR
BCR
RAMSEL
CS
IOBCR
MDACK
IORD
LTCH SDH
RSTCOUNT
COUNT
HDSEL0
HDSEL1
HDSEL2
DSEL0, 1
DSEL2, 3

231133-002


INTEL CORPORATION
TITLE 82062 SBX
SIZE B
CODE NUMBER 062PAL3.DWG
REV A
DATE SEPT 1984
SHEET 3 OF 4
231133-33
82062
74S74
74S08
74LS04
74LS14
7406
7407
74155
74LS08
510 PULLUP
330
220
WR DATA
RD CLK
RD DATA
RD GATE
DRUN
EARLY
LATE
WGEN
WCLK
INDEX
TRACK 000
SEEK COMPLETE
WRITE FAULT
READY
STEP
DIR IN
RWC
WG
STBRDY
OBRDY
BRDY
BCR
BCS
INTRQ
MR
RD
WR
CS
BDRQ
BORD
BDWR
A0-2
HDSEL0
HDSEL1
HDSEL2
DSEL2, 3
DSEL0, 1
+5V

231133-002


INTEL CORPORATION
TITLE 82062 SBX
SIZE B
CODE NUMBER 062EFRV.DWG REV A
DATE SEPT 1984 SHEET 4 OF 4
231133-34
PLL
DP8460
UK3
26LS32
UL1
26LS31
UL2
74S151
UK2
74S74
UH2
74LS174
74S04
UD1
60ns D.L.
12ns/TAP
UI2
96S02
UG2
265ns CLR
WGEN 4
WCLK 3C4
5MHz
10MHz
XTAL OSC.
OUT
UF1
20MHz
DRUN 3B1
RD GATE
RD CLK
RD DATA
WR DATA
EARLY
LATE
10MHZ
RWC
PUP1
+5V
-12 ns
+12 ns
0 ns

231133-002